VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

BLOKOVÁNÍ SLEDOVACÍCH PRVKŮ PRO PROHLÍŽEČE ZALOŽENÉ NA WEBKITGTK

TRACKER BLOCKING IN WEBKITGTK-BASED BROWSERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **SAMUEL DUDÍK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Dudík Samuel**
Program: Informační technologie
Název: **Blokování sledovacích prvků pro prohlížeče založené na WebKitGTK**
Tracker Blocking in WebKitGTK-Based Browsers
Kategorie: Web

Zadání:

1. Seznamte se s filtrovacími seznamy, tvary pravidel a blokovanými prvky.
2. Nastudujte si postup psaní rozšíření pro knihovnu WebKitGTK, seznamte se s výhodami a nevýhodami existujících rozšíření (např. wyebadblock).
3. Navrhněte rozšíření pro prohlížeče založené na knihovně WebKitGTK, které umožní blokovat sledovací prvky na základě existujících seznamů sledovacích prvků.
4. Návrh implementujte.
5. Navrhněte a implementujte spolehlivý způsob pro otestování rozšíření a porovnání s jinými existujícími rozšířeními.
6. Otestujte implementované rozšíření, implementaci porovnejte s konkurenčními řešeními.
7. Vyhodnoťte výsledky práce a navrhněte možnosti dalšího pokračování.

Literatura:

- WebKitWebExtension: WebKitGTK Reference Manual. Dostupné online na https://webkitgtk.org/reference/webkit2gtk/stable/WebKitWebExtension.html
- Imane Fouad, aj. Missed by Filter Lists: Detecting Unknown Third-Party Trackers with Invisible Pixels. PETS 2020 - 20th Privacy Enhancing Technologies Symposium, 2020, Montréal, Canada.
- Gertjan Franken, aj. Who left open the cookie jar? A comprehensive evaluation of third-party cookie policies. Proceedings of the 27th USENIX Conference on Security Symposium (SEC'18). 2018. USENIX Association, USA, str. 151-168.
- BEDNÁŘ, Martin. Automatické testování projektu JavaScript Restrictor. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Cieľom práce je vytvoriť rozšírenie pre prehliadače založené na technológii WebKitGTK, ktoré užívateľom umožňuje blokovanie reklám, sledovacích a rôznych iných nežiadúcich prvkov. Samotné rozšírenie je implementované v programovacom jazyku C. Na pozadí komunikuje so serverom napísanom v jazyku Rust, ktorý rozhoduje, či konkrétnu požiadavku zablokovať, alebo povoliť. Server využíva knižnicu adblock-rust, ktorá bola pôvodne vytvorená pre potreby prehliadača Brave. Komunikácia medzi serverom a klientom prebieha pomocou mechanizmu unixových soketov. Výsledkom práce je plnohodnotné rozšírenie určené na filtrovanie obsahu podporujúce okrem sieťového aj dynamické kozmetické filtrovanie. Súčasťou vytvoreného rozšírenia je i minimalistické GUI na jednoduchú konfiguráciu a interakciu s rozšírením.

Abstract

This thesis deals with creating an extension for WebKitGTK based browsers that allows users to block advertisements, trackers and other undesirable elements. The extension is implemented in the C programming language. It communicates with a server written in Rust that decides which requests to block and which to allow. The server uses the adblock-rust library that was developed for the Brave browser. Communication between the server and the client uses the Unix domain socket mechanism. The outcome of the thesis is a full-featured content filtering extesnion that in addition to network filtering also supports dynamic cosmetic filtering. There is also a minimalistic GUI for easy configuration and interaction with the extension.

Kľúčové slová

WebKitGTK, rozšírenie do prehliadača, filtrovanie obsahu, blokovanie reklám, sledovacie prvky

Keywords

WebKitGTK, browser extension, content filtering, ad blocking, trackers

Citácia

DUDÍK, Samuel. *Blokování sledovacích prvků pro prohlížeče založené na WebKitGTK*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Blokování sledovacích prvků pro prohlížeče založené na WebKitGTK

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Libora Polčáka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Samuel Dudík
10. mája 2021

Poďakovanie

Ďakujem vedúcemu mojej práce, Ing. Liborovi Polčákovi, Ph.D., za umožnenie pracovať na vlastnom zadaní a za prínosné konzultácie.

Obsah

Kapitola 1

Úvod

Internet bol pôvodne zameraný na zdieľanie informácií predovšetkým v akademických kruhoch. So sprístupnením jeho služieb širokej verejnosti prišla aj vlna postupne silnejúcej komercializácie. Užívatelia začali byť bombardovaní množstvom reklám na rôzne produkty a služby. Spočiatku boli reklamy neškodné a väčšinou neboli nijakým spôsobom cielené na konkrétnych užívateľov. S technologickým pokrokom a snahou o agresívnejší marketing môžme v dnešnej dobe pozorovať vysoko sofistikované metódy na zber údajov o užívateľoch a ich následne použitie pri zameriavaní reklám. Taktiež sme svedkami postupného prechodu na ekonomiku založenú na informáciách, ktorá je úzko prepojená so „Surveillance capitalism“ – ekonomický systém zameraný na komodifikáciu osobných údajov za cieľom generovať zisk. Z toho dôvodu sú rozšírenia na blokovanie požiadaviek nevyhnutné pre ochranu súkromia a anonymity na internete.

Google Chrome, prehliadač spoločnosti Google, ktorej primárnym zdrojom príjmov podľa jej výročnej správy je práve reklama [2], je dnes najpoužívanejším prehliadačom na svete s trhovým podielom viac ako 60 % [23]. S takmer 20 % je na druhom mieste prehliadač Safari vyvíjaný spoločnosťou Apple, ktorá síce má lepší prístup k súkromiu ich užívateľov, no ich produkty sú limitované iba na ich vlastnú platformu a samotná firma sa v minulosti taktiež nevyhla kontroverziám pri manipulácii s dátami užívateľov [15]. Zostávajú už len prehliadače s minoritným podielom menej ako 4 %, z ktorých časť využíva renderovacie jadro Blink, opäť vyvíjané pod záštitou spoločnosti Google. Kvôli jej zameraniu na predaj reklamného priestoru a negatívnemu postoju k súkromiu užívateľov [16], je človek, ktorý sa snaží chrániť si svoje súkromie na internete, odkázaný na jeden prehliadač – Firefox od spoločnosti Mozilla. Dopyt po alternatíve je však stále na mieste. Najmä z dôvodu nedávnych prepúšťaní zamestnancov [14] a malého trhového podielu je budúcnosť prehliadača Firefox neistá.

Možnú alternatívu predstavujú prehliadače založené na technológii WebKitGTK. Ide o port renderovacieho jadra WebKit za účelom lepšej integrácie do toolkit-u GTK. Samotný WebKit je využívaný v prehliadači Safari a je taktiež vyvíjaný spoločnosťou Apple. WebKitGTK teda benefituje zo zastrešenia veľkou spoločnosťou, no zároveň je jeho vývoj pod drobnohľadom dobrovoľníkov, ktorý okrem iného taktiež dozerajú na ochranu súkromia užívateľov.

Nevýhodou WebKitGTK je dostupnosť iba jedného rozšírenia určeného na blokovanie požiadaviek z dôvodu absencie implementácie štandardu WebExtensions. Toto rozšírenie poskytuje len veľmi obmedzenú časť funkcionality alternatívnych riešení ako napr. uBlock Origin. Medzi zásadné limitácie patrí absencia kozmetického filtrovania a užívateľského rozhrania. Cieľom tejto práce je vytvoriť rozšírenie podporujúce všetky prehliadače založené

na technológii WebKitGTK, ktoré poskytne rozsiahlejšiu funkcionalitu a vyššiu efektivitu filtrovania oproti existujúcemu riešeniu.

Kapitola 2 sa zaoberá základnou teóriou potrebnou na pochopenie zvyšku práce a analýzou najpoužívanejších filtrovacích zoznamov. Kapitola 3 popisuje existujúce riešenia na filtrovanie obsahu a demonštráciu použitia WebKitGTK API na tvorbu rozšírení. V kapitole 4 sa zaoberám návrhom riešenia. Kapitola 5 pojednáva o priebehu implementácie riešenia a problémoch, ktoré bolo nutné vyriešiť. V poslednej kapitole 6 sú popísané metódy testovania výsledného rozšírenia a jeho porovnanie s konkurenčným riešením.

Kapitola 2

Filtrovacie pravidlá, zoznamy a programy

Nasledujúca kapitola vysvetľuje základné pojmy a teóriu potrebnú k pochopeniu fungovania softvéru na blokovanie požiadaviek. Taktiež sa zaoberá analýzou najbežnejšie používaných filtrovacích zoznamov. Na konci kapitoly sú predstavené populárne filtrovacie programy na ostatných platformách.

2.1 Filtrovacie pravidlá

Ak nie je uvedené inak, všetky informácie v tejto sekcii boli čerpané z oficiálneho článku od Adblock Plus [9], resp. z jeho staršej verzie [20], a zo zhrnutia syntaxe filtrovacích pravidiel od Imre Kristoffer Eilertsen [8]. V určitých prípadoch boli informácie získané vlastným pozorovaním.

Filtrovacie pravidlo, skrátene filter, určuje či má byť konkrétna HTTP požiadavka alebo prvok na stránke zablokovaný, resp. nezablokovaný. Dokumentácia Adblock Plus [9] ich rozdeľuje na:

- **Blokovacie filtre** – Blokujú HTTP požiadavky na sieťovej úrovni.
- **Obsahové filtre** – Umožňujú skryť HTML prvky stránky. V literatúre sa taktiež objavuje výraz kozmetické filtre.
- **Výnimkové filtre** – Slúžia na negáciu iných filtrov, teda na povolenie požiadaviek zablokovaných blokovacím filtrom alebo na opätovné zobrazenie prvkov stránky, ktoré boli skryté obsahovým filtrom. Za výnimkové filtre sa označujú aj blokovacie a obsahové filtre, ktoré negujú efekt iných filtrov.

Syntax filtrovacích pravidiel

Syntax filtrovacích pravidiel nepodlieha žiadnej oficiálnej norme. Napriek tomu sa za etalón považuje syntax pravidiel Adblock Plus. Táto syntax je pri niektorých rozšíreniach obohatená o ďalšiu funkcionalitu. V ojedinelých prípadoch dochádza aj k odstráneniu funkcionality, napr. v prípade možnosti `$rewrite`, ktorá predstavovala bezpečnostné riziko a do uBlock Origin nebola nikdy implementovaná [12].

Názorné príklady rôznych typov filtrovacích pravidiel sú uvedené v obrázku 2.1.

Obr. 2.1: Príklady rôznych typov filtrovacích pravidiel. Pravidlá boli prevzaté zo zoznamu EasyList a upravené pre potreby ukážky. Snímka obrazovky bola zachytená z prostredia rozšírenia uBlock Origin za účelom získania zvýraznenia syntaxe.

Syntax blokovacích pravidiel

Hlavnou časťou blokovacích pravidiel je filtrovací vzor, ktorý slúži na nájdenie zhody v URL adrese požiadavky. Vzor je možné popísať všetkými znakmi, ktoré môžu tvoriť validnú URL adresu. Niekoľko znakov je ale rezervovaných pre špeciálny význam:

- `*` – Reprezentuje ľubovoľný (aj nulový) počet akýchkoľvek znakov.
- `^` – Reprezentuje akýkoľvek ne-alfanumerický znak (vrátane konca riadku) okrem znakov `_`, `-`, `.` a `%`.

Ak vzor začína prefixom `||`, tak doménové meno URL adresy musí začínať zvyšnou časťou vzoru (na prítomnosti prefixu `www.` nezáleží). Prefix alebo sufix `|` značí, že URL adresa musí začínať, resp. končiť týmto vzorom. V prípade, že vzor začína a zároveň končí znakom \, je vzor interpretovaný ako plnohodnotný regulárny výraz. Regexové pravidlá sú ale kvôli náročnosti vyhodnocovania používané len zriedka, resp. v špeciálnych prípadoch.

Súčasťou blokovacieho pravidla môže byť taktiež zoznam tzv. možností (angl. options). Možnosti obmedzujú platnosť pravidla iba na určitý typ požiadavky. Medzi bežne používané možnosti sa radia napr.:

- **script** – Externé skripty načítané pomocou HTML prvku `<script>`.
- **image** – Obrázky zvyčajne načítané pomocou HTML prvku `<img>`.
- **stylesheet** – Externý súbor CSS načítaný napr. pomocou HTML prvku `<link>`.
- **object** – Obsah využívaný prehliadačovými pluginmy ako napr. Java appletmi či prehrávačom Flash.
- **xmlhttprequest** – Požiadavky odoslané pomocou rozhrania XMLHttpRequest alebo fetch.
- **document** – Predstavuje samotnú webovú stránku.

- **subdocument** – Webové stránky zvyčajne načítané pomocou HTML prvku `<iframe>`.
- **third-party** – Uplatnenie pravidla obmedzuje na požiadavky iniciované z iného zdroja ako aktuálne navštívenej webovej stránky.
- **other** – Typy požiadaviek, ktoré nie sú pokryté dostupnými možnosťami, tj. ostatné typy požiadaviek.

Existujú ešte ďalšie, menej často využívané možnosti. Väčšina možností pochádza alebo má podobnosť s typmi definovanými vo WebExtensions API – webRequest.ResourceType[1]. Tie špecifikujú, o aký typ požiadavky ide, resp. v akom kontexte bola odoslaná.

Zoznam možností je od vzoru oddelený znakom doláru `$`. Jednotlivé možnosti sú od seba oddelené znakom čiarky `,`.

Špeciálnym prípadom je možnosť `domain=`, ktorá platnosť pravidla obmedzuje na zoznam domén uvedený za znakom `=`. Jednotlivé domény sú od seba oddelené znakom `|` a je možné ich negovať uvedením symbolu `~` pred doménu.

Niektoré možnosti majú alternatívne označenia, napr. možnosť `stylesheet` sa dá aplikovať skráteným tvarom `css`.

Väčšinu možností je možné negovať tj. obrátiť ich význam pridaním prefixu `~` pred možnosť. Napr. použitie negovanej možnosti `~image` pravidlo obmedzuje na všetky požiadavky okrem požiadaviek prenášajúcich súbor typu obrázok.

Syntax obsahových pravidiel

Obsahové pravidlá pozostávajú z 3 častí:

- **Zoznam domén** – Voliteľný zoznam domén, ktorých sa pravidlo týka. Jednotlivé domény sú od seba oddelené čiarkou `,`. Ak je zoznam vynechaný, tak sa pravidlo uplatňuje na všetkých doménach.
- **Separátor** – Oddeľuje zoznam domén od selektoru. Vždy pozostáva z 2 znakov `##`.
- **Selektor** – Selektor CSS určujúci, ktoré prvky stránky sú nežiadúce a majú byť skryté/odstránené.

Syntax výnimkových pravidiel

Syntax výnimkových pravidiel je takmer totožná so syntaxou blokovacích, resp. obsahových pravidiel. Odlišujú sa len v špeciálnom označení, ktorý obracia význam pravidla na výnimku.

V prípade blokovacích filtrov ide o pridanie prefixu `@@` na začiatok pravidla. Pri obsahových filtrov je pôvodný separátor nahradený reťazcom `#@#`.

2.2 Filtrovacie zoznamy

Filtrovací zoznam je zoznam filtrovacích pravidiel. Filtrovacie zoznamy nepodliehajú žiadnemu oficiálnemu deleniu. Väčšinou sú zamerané na určitú oblasť záujmu, napr. geografický

[1] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/ResourceType

región[2] či konkrétna stránka/spoločnosť[3]. Existujú aj také zoznamy, ktoré sa snažia byť univerzálne a pokryť čo najväčšie množstvo generický reklám či sledovacích prvkov. Vybrané univerzálne zoznamy sú detailnejšie popísané v sekcii 2.2.

Analýza vybraných filtrovacích zoznamov

Zmyslom analýzy je získanie informácií potrebných na lepšie pochopenie nárokov kladených na filtrovacie rozšírenia. Výsledok analýzy je taktiež možné použiť pri testovaní výsledného riešenia na vytvorenie podmienok simulujúcich bežné použitie.

Analýza a rozdelenie pravidiel prebiehalo na základe rozsiahlej dokumentácie filtrovacích pravidiel [8] od Imre Kristoffer Eilertsen, ktorý sa profesionálne podieľa na tvorbe filtrovacích zoznamov pre rozšírenie Adblock Plus.

Skript určený na analýzu filtrovacích zoznamov bol implementovaný v skriptovacom jazyku Python. Výsledný graf bol vygenerovaný pomocou knižnice seaborn.

Konkrétne som sa zameral na nasledujúcich 9 zoznamov:

- **EasyList**[4] – Univerzálny zoznam pôvodne vytvorený pre Adblock.
- **EasyPrivacy**[5] – Doplnkový zoznam pre EasyList zameraný na ochranu súkromia a osobných údajov užívateľov.
- **uBlock filters**[6] – Filtre optimalizované pre uBlock, dopĺňajúce Easylist.
- **uBlock filters – Privacy**[7] – Zoznam špeciálne určený pre uBlock Origin, zameraný na ochranu súkromia a osobných údajov užívateľov.
- **uBlock filters – Badware risks**[8] – Účelom je upozorniť užívateľa na webové stránky, ktoré sa v minulosti dopustili šírenia nežiadúceho softvéru.
- **uBlock filters – Resource abuse**[9] – Zamerané na blokovanie webových stránok, ktoré zneužívajú prostriedky počítača bez informovaného súhlasu užívateľa.
- **uBlock filters – Unbreak**[10] – Zoznam špeciálne určený na „opravenie“ stránok znefunkčnených použitím predvolených zoznamov tretích strán pre uBlock Origin.
- **Online Malicious URL Blocklist**[11] – Zoznam pozostáva zväčša z IP adries a doménových mien distribujúcich škodlivé programy.
- **Peter Lowe's Ad and tracking server list**[12] – Univerzálny zoznam pozostávajúci exkluzívne z doménových mien.

[2] napr. https://raw.githubusercontent.com/tomasko126/easylistczechandslovak/master/filters.txt
[3] napr. https://secure.fanboy.co.nz/fanboy-antifacebook.txt
[4] https://easylist.to/easylist/easylist.txt
[5] https://easylist.to/easylist/easyprivacy.txt
[6] https://cdn.jsdelivr.net/gh/uBlockOrigin/uAssets@master/filters/filters.txt
[7] https://raw.githubusercontent.com/uBlockOrigin/uAssets/master/filters/privacy.txt
[8] https://raw.githubusercontent.com/uBlockOrigin/uAssets/master/filters/badware.txt
[9] https://raw.githubusercontent.com/uBlockOrigin/uAssets/master/filters/resource-abuse.txt
[10] https://cdn.statically.io/gh/uBlockOrigin/uAssets/master/filters/unbreak.txt
[11] https://curben.gitlab.io/malware-filter/urlhaus-filter-online.txt
[12] https://pgl.yoyo.org/adservers/serverlist.php?hostformat=hosts&showintro=1&mimetype=plaintext

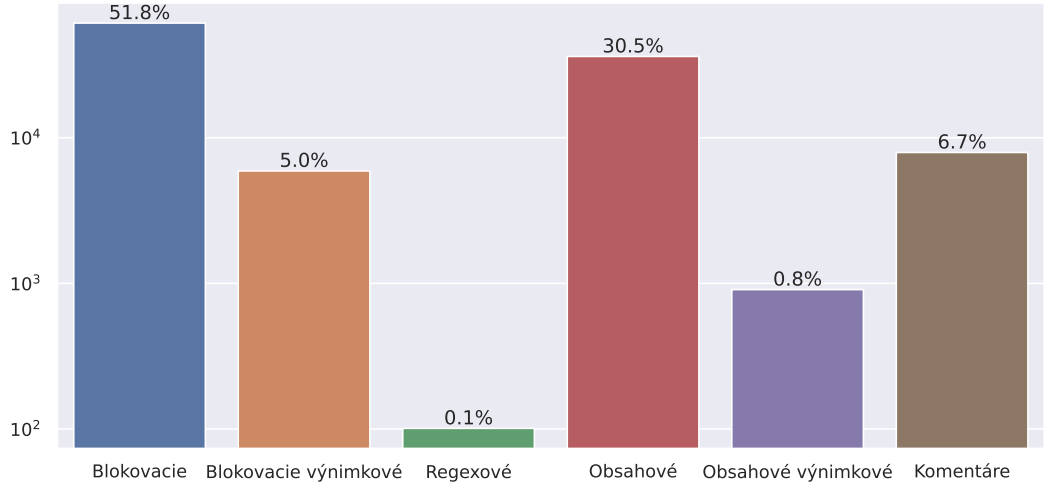
Tieto zoznamy pokrývajú všetku bežnú funkcionalitu požadovanú od blokovača požiadaviek. Po novej inštalácii uBlock Origin sú automaticky aktivované. Z toho dôvodu verne reprezentujú použité zoznamy priemerného užívateľa, ktorý nevykonal žiadne manuálne úpravy konfigurácie.

Výsledky analýzy sú prezentované v tabuľke 2.1. Tie isté výsledky sú dostupné aj vo forme grafu na obrázku 2.2, z ktorého je možné vyčítať aj percentuálne hodnoty.

Tabuľka 2.1: Počet filtrov podľa typu.

	Všetky	Blokovacie	Blokovacie W	Regex	Obsahové	Obsahové W	K
EL	61 596	31 804	1 864	71	26 750	731	340
EP	18 439	17 280	817	20	0	0	315
UF	24 290	2 284	2 220	3	9 269	90	5 627
UFP	203	53	4	2	10	1	92
UFB	483	244	0	1	33	0	137
UFR	256	86	0	0	18	0	84
UFU	3 944	377	1 012	4	74	84	1 304
OM	5 591	5 585	0	0	0	0	6
PL	3 569	3 555	0	0	0	0	14
Spolu	118 371	61 268	5 917	101	36 154	906	7 919

[W] Výnimkové filtre
[K] Komentáre a informačné riadky
[EL] EasyList
[EP] EasyPrivacy
[UF] uBlock filters
[UFP] uBlock filters – Privacy
[UFB] uBlock filters – Badware risks
[UFR] uBlock filters – Resource abuse
[UFU] uBlock filters – Unbreak
[OM] Online Malicious URL Blocklist
[PL] Peter Lowe's Ad and tracking server list



Obr. 2.2: Graf vyjadrujúci podiel počtu pravidiel vo všetkých analyzovaných zoznamoch podľa typu filtra.

2.3 Filtrovacie programy

Filtrovacie programy, na základe filtrovacích zoznamov, filtrujú požiadavky a prvky stránok. Bez filtrovacích zoznamov by ich správne fungovanie nebolo možné. Filtrovacie programy majú väčšinou podobu rozšírení do prehliadačov. Medzi najznámejšie rozšírenia patrí uBlock Origin či Adblock Plus.

Adblock Plus

Adblock Plus bolo jedno z prvých rozšírení umožňujúce blokovať reklamy v prehliadači. Vzniklo ako vylepšená verzia pôvodného Adblock-u a do dnešného dňa si na základe počtu stiahnutí stále drží miesto najpoužívanejšieho rozšírenia určeného na blokovanie reklám v prehliadačoch Firefox a Google Chrome.

V roku 2011 čelila firma vyvíjajúca Adblock Plus kritike kvôli novému programu „Acceptable Ads“ [19], ktorý pri predvolených nastaveniach rozšírenia povolil zobrazovanie „menej rušivých“ reklám. Členstvo v programe je pre firmy, ktorých reklamy uvidí vďaka programu mesačne viac ako 10 miliónov užívateľov, spoplatnené. Firmy, ktoré túto hranicu neprekročia, môžu členstvo užívať zadarmo. Na základe tohto programu je možné predpokladať, že za rozšírením Adblock Plus stojí komerčná entita, ktorej hlavným záujmom je zisk.

uBlock Origin

V roku 2014 sa objavilo rozšírenie uBlock Origin s cieľom priniesť efektívnejšie filtrovanie. Oproti predošle spomínaným programom nie je uBlock Origin zaštítený žiadnou komerčnou spoločnosťou. V README.md projektu je uvedené, že finančné dary nie sú potrebné [13]. README.md ďalej odkazuje na filozofiu a manifesto projektu, ktoré užívateľov presviedča, že blokovanie reklám nie je v žiadnom prípade krádež a že jedine užívateľ môže rozhodovať o tom, aký obsah je v jeho prehliadači povolený.

Filtrovací program prehliadača Brave

Prehliadač Brave taktiež disponuje filtrovacím mechanizmom, no od Adblock Plus alebo uBlock Origin sa odlišuje tým, že v podstate nejde o klasické rozšírenie do prehliadača. Ide o samostatný program, ktorý je napísaný v programovacom jazyku Rust a z prehliadača Brave je volaný pomocou mechanizmu „Foreign function interface“[13]. Napriek tomu funkcionalita filtrovacieho programu stále závisí na filtrovacích zoznamoch.

V minulosti Brave čelil dvom kontroverziám. Prvou z nich bolo vkladanie tzv. kódov „affiliate“ do niektorých URL adries odkazujúcich na obsah o kryptomenách [17]. Okrem toho v minulosti vyberal dary v mene tvorcov obsahu, ktorí o tejto činnosti neboli dopredu oboznámení a v určitých prípadoch si Brave následne nárokoval právo na ponechanie týchto darov [10]. Oba problémy boli po kritike vyriešené.

[13] https://github.com/brave/adblock-rust-ffi

Kapitola 3

Technológia WebKitGTK

WebKitGTK je port renderovacieho jadra WebKit, integrovaný do widget toolkit-u GTK. Je oficiálnym renderovacím jadrom platformy GNOME a umožňuje vytvárať plnohodnotné webové prehliadače s podporou technológií HTML, CSS, Javascript, ale aj WebRTC či GStreamer [25].

3.1 Dostupné webové prehliadače

Prehliadačov založených na technológii WebKitGTK, ktoré by teda mohli benefitovať z dostupnosti nového rozšírenia na filtrovanie obsahu, existuje spolu viac ako 13. Zväčša ide o menšie experimentálne prehliadače snažiace sa o inováciu, ale aj o pomerne populárne aplikácie s väčším počtom užívateľov. Informácie o veľkosti užívateľskej základne dostupné nie sú, no približnú predstavu môže poskytnúť spoločný počet hviezd na platforme Github, ktorým užívatelia môžu vyjadriť záujem o daný repozitár. Ten sa pohybuje na úrovni viac ako 9 000 hviezd. Zoznam väčšiny dostupných prehliadačov rady WebKitGTK je dostupný na Arch Wiki[1].

Nasledujúce podkapitoly sa venujú jednotlivým najpoužívanejším WebKitGTK prehliadačom a ich unikátnej funkcionalite, ktorou sa odlišujú od konkurenčných riešení.

Nyxt

Nyxt[2] je prehliadač zameraný na ovládanie pomocou klávesnice. Je inšpirovaný textovými editormi Emacs a Vim, s ktorými zdieľa niektoré klávesové skratky. Sústreďuje sa teda skôr na pokročilých užívateľov. Je implementovaný v programovacom jazyku Lisp, vďaka čomu môže byť časť jeho zdrojového kódu upravovaná za behu.

vimb

Prehliadač vimb[3] s prehliadačom Nyxt zdieľa viaceré vlastnosti. Je taktiež zameraný na pokročilých užívateľov a vo veľkej miere na ovládanie pomocou klávesnice. Vimb sa však snaží byť čo najviac verný textovému editoru vim. Po jeho vzore poskytuje modálne užívateľské rozhranie, totožné klávesové skratky a viaceré registre na kopírovanie a vkladanie textu.

[1] https://wiki.archlinux.org/index.php/List_of_applications/Internet
[2] https://nyxt.atlas.engineer/
[3] https://fanglingsu.github.io/vimb/

Konfigurácia nastavení dokonca využíva rovnakú syntax. Snahou vimb je teda čo najväčšie uľahčenie prepínania mentálneho kontextu medzi textovým editorom a prehliadačom.

surf

Webový prehliadač surf[4], vyvíjaný zoskupením suckless, sa od konkurenčných riešení odlišuje zameraním sa na v niektorých prípadoch až extrémny minimalizmus a jednoduchosť použitia. Zámerne nepodporuje funkcionalitu viacerých kariet. Tú je možné do prehliadača pridať použitím separátne nástroja tabbed vyvíjaného rovnakým tímom. Grafické užívateľské prostredie je implementované využitím ďalšieho nástroja oddeleného od prehliadača, ktorý je opäť vyvíjaný zoskupením suckless. Samotný prehliadač je implementovaný v menej ako 2 200 riadkoch jazyka C. Z manifestu zoskupenia a zo smeru vývoja ich nástrojov teda možno dedukovať silný vplyv Unixovej filozofie, ktorá uprednostňuje jednoduchý softvér zameraný na riešenie jedného problému, čo uľahčuje spoluprácu s inými nástrojmi [18].

GNOME Web

GNOME Web, v minulosti tiež známy ako Epiphany, je prehliadač vyvíjaný komunitou „The GNOME Project“. Ide o konvenčný prehliadač disponujúci podobnou funkcionalitou ako prehliadače populárne u bežných užívateľov (napr. Google Chrome, Firefox). GNOME Web ako jeden z prvých WebkitGTK prehliadačov umožnil užívateľom blokovať sledovacie prvky a reklamy. Implementácia blokovača sa neustále mení a vyvíja, ale v čase písania práce je celé fungovanie blokovača založené na API filtrovania obsahu (content filtering). Nevýhodou tejto API je neštandardný formát filtrovacích zoznamov, ale predovšetkým obmedzenie na maximálne 50 000 aktívnych filtrov. Filtrovanie požiadaviek je súčasťou samotného prehliadača a nie je možné ho jednoducho použiť v iných prehliadačoch. Problematike Content blocker API sa dôkladnejšie venujem v sekcii 3.2.

Integrované webové prehliadače

Okrem klasických webových prehliadačov môže byť vykresľovacie jadro WebKitGTK integrované i do iných aplikácií ako napr. čítačiek RSS či emailových klientov. Zoznam aplikácií s rôznym zameraním využívajúcich jadro WebKitGTK je dostupný v rámci oficiálnej WebKit Wiki[5]. Kompatibilita týchto aplikácií s rozšíreniami pre WebKitGTK, vrátane môjho riešenia, je podmienená nastavením priečinku na ukladanie rozšírení pomocou funkcie `webkit_web_context_set_web_extensions_directory` v danej aplikácii. Informácia o kompatibilite s rozšíreniami v dokumentácii často chýba, a tak je nutné spomínanú funkciu nájsť v zdrojovom kóde aplikácie a následne z nej odvodiť cestu k priečinku, do ktorého má byť rozšírenie umiestnené. Napr. v prípade čítačky RSS Liferea[6] je z jej zdrojového kódu[7] možné cestu k priečinku odvodiť na `/usr/lib/liferea/web-extension`.

[4] https://surf.suckless.org/

[5] https://trac.webkit.org/wiki/ApplicationsGtk

[6] https://lzone.de/liferea/

[7] https://github.com/lwindolf/liferea/blob/32983496a530875c015a9d9a06c5cfeedf0ce0b4/src/webkit/webkit.c#L379

3.2 Existujúce riešenia na filtrovanie obsahu

Napriek pomerne rozsiahlym možnostiam tvorby rozšírení pre prehliadače založené na technológii WebkitGTK, je dostupné len 1 rozšírenie umožňujúce blokovanie reklám a sledovacích prvkov. Táto podkapitola sa zaoberá práve týmto rozšírením, ale taktiež inými dostupnými riešeniami na filtrovanie obsahu.

Content blocker API

Webkit v roku 2015 predstavil Content blocker API ako reakciu na rozšírenia určené na blokovanie obsahu. Dostupné rozšírenia boli kritizované najmä za problémy týkajúce sa výkonu, vyššej spotreby batérie a pomalého načítavania stránok, ktoré boli odôvodnené predovšetkým použitím Javascriptu ako programovacieho jazyka [21]. Content blocker API nie je kompatibilná s Adblock Plus filter syntaxou a prichádza s novým štrukturovaným formátom na deklaratívne ukladanie filtrov, ktoré sú po načítaní zkompilované do efektívnejšieho bajtkódu (angl. bytecode). Ten je následne spúšťaný nad každou požiadavkou.

Táto API je dostupná taktiež vo WebKitGTK, no jej použitie je limitované:

- Počet aktívnych filtrov je obmedzený na 50 000.
- Neumožňuje dynamické zakázanie/povolenie filtrov.
- Prístup k API nie je možný z prostredia rozšírenia.

Podľa analýzy filtrovacích zoznamov v sekcii 2.2 tvoria predvolené zoznamy uBlock Origin spolu 110 452 filtrov. Stanovený limit na 50 000 filtrov teda ani z polovice nepokrýva základnú sadu filtrov.

Najväčší problém predstavuje dostupnosť rozhrania. Content blocker API vyžaduje prístup k objektu WebView, ktorý ale z rozšírenia dostupný nie je. Z toho dôvodu je použitie limitované na implementáciu priamo v samotnom prehliadači, čím prichádzame o všetky benefity univerzálneho rozšírenia.

wyebadblock

Wyebadblock je jediné skutočne externé rozšírenie pre WebkitGTK prehliadače. Na filtrovanie nevyužíva content filtering API, ale vlastnú implementáciu, ktorá bola z väčšej časti prevzatá zo staršej verzie prehliadača GNOME Web. Kód bol upravený pre kompatibilitu s ďalšími WebkitGTK prehliadačmi. Taktiež prináša menšiu spotrebu zdrojov vďaka zdieľaniu filtrovacieho serveru medzi klientmi. Samotné filtrovanie je založené na využívaní hašovacích tabuliek knižnice Glib. V prípade, že užívateľ nepoužíva GNOME Web, je wyebadblock prakticky jediné rozšírenie, ktoré môžu užívatelia WebkitGTK prehliadačov použiť na blokovanie reklám. Rozšírenie trpí na tie isté problémy ako verzia blokovača staršej verzie GNOME Web, z ktorej bola väčšina kódu prevzatá [1]:

- Absencia dynamického kozmetického filtrovania.
- Nepodporuje viacero filtrovacích možností.
- Neefektívnosť filtrovania (napr. zbytočné používanie regulárnych výrazov pri kontrole, ktoré možnosti sú aktívne).

Súbor hosts

Poslednou možnosťou filtrovania obsahu je použitie súboru hosts, ktorý slúži na lokálny preklad hostiteľských (angl. hostname) a doménových mien na IP adresy. Tento súbor sa na Unixových systémoch nachádza v zložke `/etc/`. Syntax, ktorú vyžaduje, je ukázaná vo výpise 3.1.

```
#<ip-address> <hostname.domain.org> <hostname>
127.0.0.1 localhost.localdomain localhost
```

Výpis 3.1: Ukážka predvoleného obsahu súboru `/etc/hosts`.

Každý riadok začína IP adresou oddelenou bielym znakom (napr. medzera alebo tabulátor) od 1 alebo viacerých hostiteľských mien, ktoré majú byť prekladané na zadanú IP adresu. To je možné využiť na presmerovanie nežiadúcich doménových mien na IP adresu `0.0.0.0`, prípadne na `localhost` (`127.0.0.1`). Obe IP adresy majú špeciálny význam, no v kontexte blokovania je dôležitý fakt, že na zariadení znemožňujú načítanie špecifikovaného doménového mena. Toto obmedzenie sa vzťahuje na celý operačný systém a je teda nezávislé od použitej aplikácie. Napr. výpis 3.2 znemožní navštívenie webovej stránky nachádzajúcej sa na adrese `example.com` a teda aj načítanie všetkých jej ciest.

```
0.0.0.0 example.com
```

Výpis 3.2: Pravidlo, ktoré po umiestení do súboru `/etc/hosts` zabráni načítaniu obsahu z domény `example.com`.

Tento postup je kvôli nedostatku rozšírení na blokovanie požiadaviek populárny práve u užívateľov prehliadačov WebKitGTK. Napr. autori prehliadača surf na ich oficiálnej stránke uvádzajú návod[8] na použitie súboru hosts. Nevýhodou filtrovania obsahu pomocou súboru hosts je úplná absencia kozmetického filtrovania. Okrem toho sa presmerovanie vzťahuje len na doménové meno a nie je tak možné špecifikovať konkrétnu cestu webovej stránky, ktorá má byť zablokovaná. Užívateľ tak môže zablokovať len celú webovú stránku, alebo nič. Ďalší problém predstavuje použitie väčšieho počtu mapovacích pravidiel, ktoré môže mať negatívny vplyv na rýchlosť prehliadania.

3.3 Tvorba rozšírení na filtrovanie obsahu

Informácie v nasledujúcej sekcii boli prevzaté z oficiálnej dokumentácie WebKitGTK [11].

Rozšírenia pre prehliadače WebKitGTK je nutné písať v programovacom jazyku C. Šírené sú formou dynamickej knižnice (koncovka `.so` pre Linux a `.dll` pre Windows), ktorá je načítaná až za behu samotnej aplikácie. Neobsahujú funkciu `main`, ale inicializačnú funkciu `webkit_web_extension_initialize`, ktorá je zavolaná ihneď po načítaní rozšírenia webovým procesom aplikácie/prehliadača. Pre správne fungovanie je knižnicu potrebné uložiť do zložky, ktorú si každý prehliadač špecifikuje sám pomocou funkcie `webkit_web_context_set_web_extensions_directory`. Táto funkcia musí byť zavolaná ako prvá v danom kontexte `WebKitWebContext`, ideálne zachytením signálu `initialize-web-extensions` pomocou funkcie `g_signal_connect`. Táto funkcia sa pripojí na špecifikovaný signál. Jej argumentom je predaná funkcia callback, ktorá bude zavolaná pri nastatí uda-

[8] https://surf.suckless.org/files/adblock-hosts/

losti identifikovanej daným signálom. WebkitGTK sprístupňuje nízko-úrovňovú API, ktorá poskytuje 3 hlavné funkcie [5]:

- Prístup k DOM (Document Object Model).
- Signál `send-request`, ktorý umožňuje upraviť akúkoľvek požiadavku na server ešte pred jej odoslaním, prípadne ju nikdy neodoslať.
- Vykonanie vlastného Javascript-u.

Pre účely Bakalárskej práce budú najdôležitejšie práve práca so signálom `send-request` a možnosť spúšťania vlastného Javascript-u.

Demonštrácia filtrovania požiadaviek a prvkov stránky

V rámci demonštrácie predstavím časti WebkitGTK API, ktoré sú kľúčové pre správne fungovanie blokovača požiadaviek.

V tele inicializačnej funkcie `webkit_web_extension_initialize` je najprv nutné pripojiť funkciu callback k signálu `page-created`, ktorý je zavolaný pri každom vytvorení objektu `WebKitWebPage`. Ten reprezentuje webovú stránku, ku ktorej signálom sa budú pripájať ďalšie funckie callback.

Pripojenie funkcie callback k signálu je možné pomocou funkcie `g_signal_connect`. Konkrétne použitie pre signál `page-created` je uvedené vo výpise 3.3.

```
G_MODULE_EXPORT void webkit_web_extension_initialize(WebKitWebExtension *
    extension)
{
    g_signal_connect(extension, "page-created", G_CALLBACK(
        web_page_created_callback), NULL);
}
```

Výpis 3.3: Ukážka pripojenia funkcie callback k signálu `page-created`.

Blokovanie požiadaviek

Signál `send-request` je zavolaný vždy pred zaslaním požiadavky na server. Umožňuje požiadavku modifikovať, alebo úplne zabrániť v jej odoslaní. Vo `web_page_created_callback` sa na tento signál pripája funkcia `web_page_send_request` opäť pomocou funkcie `g_signal_connect`.

V tele funckie `web_page_send_request`, ktorá bude vďaka pripojeniu na signál zavolaná pred odoslaním každej požiadavky, je už možné pristupovať k informáciám o požiadavkách pomocou funkcií:

- `webkit_web_page_get_uri` – Vracia doménové meno spolu s použitým protokolom.
- `webkit_uri_request_get_uri` – Okrem doménového mena a protokolu vracia aj cestu k súboru s parametrami, tj. vracia celú URL adresu.

Tieto funkcie je možné použiť pri rozhodovaní, či konkrétna požiadavka má byť odoslaná na server, alebo či má byť zahodená. Toto rozhodnutie sa programu predáva návratovou

hodnotou funkcie `web_page_send_request`. Návratová hodnota `true` vyvolá zahodenie požiadavky.

Vyššie predstavené funkcie pokrývajú všetku funkcionalitu potrebnú na klasické blokovanie požiadaviek, ktoré na základe analýzy v sekcii 2.2 predstavuje najväčšiu časť blokovania.

Obsahové filtrovanie

Obsahové filtrovanie, resp. kozmetické filtrovanie, je závislé na možnosti manipulácie HTML DOM. Do verzie 2.21.92 bola na prístup k HTML dokumentu odporúčaná natívna C API `WebKitDOMDocument`. Tá sa od verzie 2.22 stala zastaranou a bola nahradená novým API rozhraním `JavaScriptCore` umožňujúcim volanie vlastného kódu Javascript vloženého do webovej stránky.

Manipulovať s dokumentom DOM je možné až po jeho načítaní. V tele funkcie `web_page_created_callback` je nutné vytvorenie pripojenia funkcie `document_loaded_callback` na signál `document-loaded`, ktorý funkciu zavolá po načítaní DOM dokumentu objektu `WebKitWebPage`.

V tele funkcie `document_loaded_callback` je už možné pristúpiť k Javascript kontextu webovej stránky. Pomocou kódu vo výpise 3.4 je cez tento kontext možné zavolať akýkoľvek kód v jazyku Javascript.

```
JSCContext *js_context = webkit_frame_get_js_context(frame);
JSCValue *value = jsc_context_evaluate(js_context, "document.querySelector
    ('#logo_homepage_link').remove()", -1);
```

Výpis 3.4: Ukážka volania kódu jazyka Javascipt cez kontext webovej stránky.

Kód Javascript `document.querySelector('#logo_homepage_link').remove()` v HTML dokumente vyhľadá element s identifikátorom `logo_homepage_link`, ktorý je následné odstránený. Vo svojej podstate ide o jednoduchú formu kozmetického filtrovania bez komplexnejšej logiky.

Kapitola 4

Návrh riešenia na filtrovanie obsahu

Cieľom tejto kapitoly je predstaviť algoritmy a prístupy použité v existujúcich riešeniach, ktoré budú použité v implementačnej časti práce.

Ako bolo spomenuté v kapitole 3.2, jediným konkurenčným riešením v oblasti blokovania obsahu pre prehliadače WebKitGTK, je rozšírenie wyebadblock. Toto rozšírenie neposkytuje žiadnu formu kozmetického filtrovania a nedisponuje žiadnym užívateľských rozhraním. Z toho dôvodu je pri analýze užívateľských potrieb nutné siahnuť po riešeniach na iných platformách.

Medzi najdôležitejšiu funkcionalitu navrhovaného rozšírenia na blokovanie požiadaviek patrí:

- **Efektívnosť blokovacej logiky** – Ako rýchlo a s akým využitím výpočtových zdrojov je algoritmus schopný rozhodovať, či má byť konkrétna požiadavka/prvok zablokovaný, alebo nie.
- **Rozsah podporovanej syntaxe** – Aké množstvo typov pravidiel program podporuje. Úzko súvisí s efektívnosťou.
- **Prívetivosť užívateľského rozhrania** - Jeho jednoduchosť a intuitívnosť použitia.

Pre novovytvorené rozšírenie som zvolil názov „BlocKit“.

4.1 Filtrovacia logika

Pri návrhu filtrovacej logiky som sa zameral na analýzu technológií použitých v už existujúcich riešeniach.

Bloomov filter

Všetky informácie v nasledujúcej podsekcii boli prevzaté z článku B. H. Blooma [3], publikovanom vo vedeckom časopise „Communications of the ACM“.

Bloomov filter je pravdepodobnostná dátová štruktúra slúžiaca na rýchle a pamäťovo efektívne otestovanie príslušnosti daného prvku do množiny. Neuchováva presnú reprezentáciu jednotlivých prvkov množiny a tak prítomnosť prvku v množine je schopná určiť len s určitou pravdepodobnosťou. Avšak, absenciu prvku určuje so 100 % istotou.

Princíp jej fungovania spočíva v použití bitového poľa. Pri pridaní nového prvku do množiny je prvok predaný jednej alebo viacerým hašovacím funkciám. Výstupné hodnoty predstavujú indexy bitového poľa, ktoré sú následne nastavené na hodnotu 1.

Pri overení príslušnosti prvku do množiny sú z testovaného prvku opäť vypočítané haše, ktoré reprezentujú indexy bitového poľa. Pokiaľ sú všetky bity na vrátených indexoch rovné 1, prvok sa v množine nachádza s určitou pravdepodobnosťou. Ak je ale čo i len jeden bit nastavený na hodnotu 0, je možné prehlásiť, že sa prvok v množine určite nenachádza.

Skoršia verzia filtrovacieho jadra prehliadača Brave Bloomov filter využívala na rýchle vyradenie žiadúcich požiadaviek z filtrovacieho procesu. Použitie tejto dátovej štruktúry bolo odôvodnené pozorovaním, z ktorého vyplývalo, že väčšina požiadaviek je žiadúca.

Neskôr bolo na základe ďalšej analýzy zistené, že prevaha žiadúcich požiadaviek nie je až taká značná ako sa pôvodne predpokladalo. V snahe nájsť optimálnejšie riešenie sa tím Brave rozhodol podľa vzoru rozšírení uBlock Origin a Ghostery prejsť na blokovaciu logiku založenú na tokenizácii.

Tokenizácia

Celý proces tokenizácie bol prevzatý z článku oznamujúcom novú blokovaciu logiku prehliadača Brave [4].

Tokenizácia funguje vďaka faktu, že akýkoľvek podreťazec pravidla, musí byť taktiež obsiahnutý v akejkoľvek URL požiadavky, s ktorou sa pravidlo zhoduje. Podreťazcom sa myslí ľubovoľný alfanumerický reťazec, ktorý je súčasťou filtrovacieho pravidla a od zvyšku reťazca je oddelený separátorom (napr. bodka, lomka atď.). Pre tieto podreťazce sa ustálilo pomenovanie tokeny. Výnimkou sú regexové pravidlá a podreťazce, ktoré začínajú alebo končia špeciálnym znakom *.

Pred hlavným spracovaním filtrovacích pravidiel, je najprv potrebné vytvoriť histogram vyjadrujúci počet výskytov tokenov obsiahnutých vo všetkých pravidlách.

Každé pravidlo, ktoré sa nachádza v použitých filtrovacích zoznamoch, je nasledujúcim spôsobom spracované:

1. Pravidlo je rozdelené na tokeny.
2. Z každého tokenu je vypočítaný haš.
3. Ku každému tokenu je pridelený jeho počet výskytov vo všetkých pravidlách.
4. Pravidlo je pridané do zoznamu indexovaného tokenom, ktorý sa v ostatných pravidlách vyskytuje najmenej často, tj. je najdiskriminatívnejší.

Výsledkom spracovania je dátová štruktúra zhlukujúca pravidlá do skupín, ktoré zdieľajú spoločný najdiskriminatívnejší token. Táto štruktúra je následne použitá na rýchle zúženie množstva potenciálnych pravidiel, s ktorými by sa testovaná požiadavka mohla zhodovať.

Pri kontrole požiadavky je jej URL adresa opäť rozdelená na tokeny, ktoré sú následne zahašované. Tokeny sú po jednom testované:

1. Podľa tokenu je vyhľadaný jeho zhluk pravidiel.
2. Každé pravidlo zhluku je porovnávané na zhodu s testovanou požiadavkou.
3. V prípade zhody je kontrola predčasne ukončená a požiadavka je zablokovaná.

4. Ak po prejdení celého zhluku pravidiel nie je nájdená žiadna zhoda, je vybraný nový zhluk podľa nasledujúceho tokenu a algoritmus sa vracia na krok 1.

5. Ak už boli skontrolované všetky tokeny URL adresy, požiadavka nie je zablokovaná.

V prípade, že je požiadavka označená za nežiadúcu, je pravidlo ešte raz skontrolované vyššie uvedeným algoritmom nad zoznamom výnimkových filtrov. V prípade nájdenia zhody je nežiadúci stav požiadavky anulovaný a je prepustená ďalej.

Dokumentácia knižnice adblock-rust obsahuje vizualizáciu vyššie uvedeného algoritmu pre lepšie pochopenie[1].

Keďže cieľom práce je vytvoriť funkčné rozšírenie na filtrovanie obsahu pre platformu WebKitGTK, najvhodnejším návrhom filtrovacej logiky je použitie osvedčeného prístupu s použitím tokenizácie.

4.2 Užívateľské rozhranie

Keďže hlavnou funkcionalitou blokovača požiadaviek je automatické filtrovanie obsahu, mal by byť za bežných okolností pre užívateľa neviditeľný. Užívateľ by nemal byť nútený k manuálnej interakcií a zobrazovanie informácií o samotnom filtrovaní by taktiež malo byť minimalizované. Pod bežnými okolnosťami je myslená doba, počas ktorej žiadne navštívené stránky nevykazujú znaky nefunkčnosti, ktoré by mohli byť spôsobené filtrovaním a teda nevyžadujú manuálny zásah užívateľa. Takéto situácie nastávajú pri použití primeraného počtu vhodných filtrovacích zoznamov zriedka. Z toho dôvodu by malo byť užívateľské rozhranie blokovača skryté za ikonou, prípadne klávesovou skratkou, po ktorých aktivovaní by sa zobrazili ďalšie ovládacie prvky.

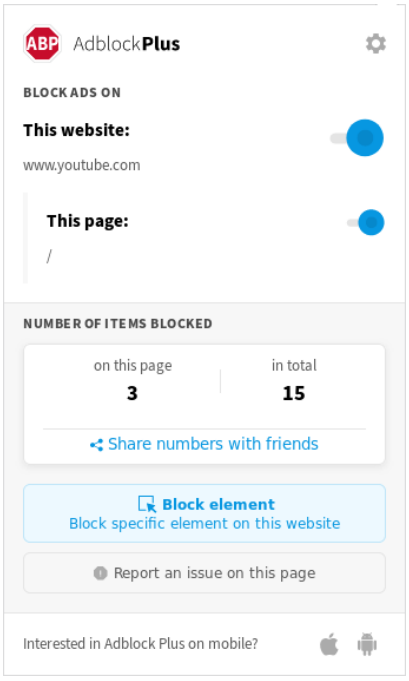
Od hlavného užívateľského rozhrania bude najčastejšie vyžadované:

- Vypnutie/zapnutie filtrovania pre aktuálne navštívenú stránku.
- Manuálne zablokovanie prvku stránky.
- Prístup k zoznamu zablokovaných požiadaviek v prípade nefunkčnej stránky.
- Prístup k nastaveniam.

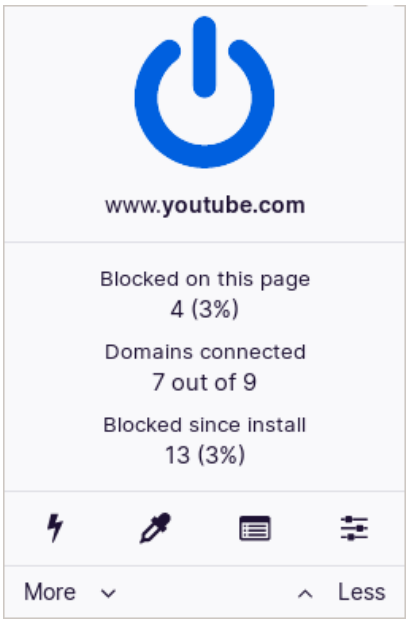
GUI rozšírenia Ghostery na obrázku 4.3 sa nejaví ako príliš vhodné. Je zbytočne veľké, neprehľadné a niektorá funkcionalita nie je dostupná z hlavnej karty.

GUI rozšírení Adblock Plus (obrázok 4.1) a uBlock Origin (obrázok 4.2) sú už dostatočne prehľadné a všetka potrebná funkcionalita je dostupná jedným kliknutím. Jediným rozdielom je konzistentnejší a viac minimalistický vzhľad rozhrania rozšírenia uBlock Origin. Vzhľadom na užívateľskú základňu alternatívnych prehliadačov je tento dizajn preferovaný a z toho dôvodu bude pri implementácii použitý ako referenčný.

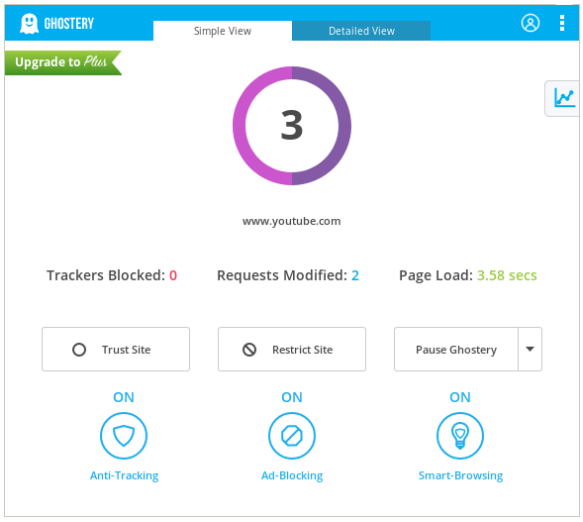
[1] https://raw.githubusercontent.com/brave/adblock-rust/master/docs/algo.png



Obr. 4.1: GUI rozšírenia AdBlock Plus.



Obr. 4.2: GUI rozšírenia uBlock Origin.



Obr. 4.3: GUI rozšírenia Ghostery.

Kapitola 5

Implementácia

Nasledujúca kapitola sa zaoberá celým procesom vývoja riešenia na filtrovanie obsahu. Prvá sekcia sa venuje implementácii blokovacej logiky v jazyku C, ktorá bola v polovici vývoja zanechaná a nahradená serverom napísanom v jazyku Rust s využitím knižnice adblock-rust. Ďalšie sekcie sa teda zaoberajú dôvodom na zmenu architektúry, spôsobom využitia knižnice a implementáciou samotného serveru a klienta – rozšírenia. Posledná sekcia popisuje výsledné užívateľské rozhranie a nástroje, ktoré boli pri jeho tvorbe použité.

5.1 Blokovacie jadro v jazyku C

Blokovacie jadro malo byť pôvodne napísané v programovacom jazyku C z dôvodu natívnej dostupnosti WebKitGTK API, ktorá je využitá na implementáciu rozšírenia, z ktorého by bolo jadro volané. Samozrejme, existujú možnosti prepojenia s inými jazykmi (angl. language binding) vyvíjané tretími stranami, ktoré ale na pozadí stále volajú natívne C funkcie WebKitGTK API. Tento prístup, výmenou za vyššiu úroveň abstrakcie a zjednodušenie programovania, so sebou prináša ďalší bod potenciálneho zlyhania v podobe spoliehania sa na knižnicu vyvíjanú neoficiálnym vývojárom.

Implementácia blokovacieho jadra spočívala v prispôsobení algoritmu popísaného v sekcii 4.1 potrebám prostredia WebKitGTK. Pri vývoji boli použité pomocné knižnice Glib[1] a GIO[2]. Knižnica GIO sprístupňuje vrstvu vysokej abstrakcie nad súborovým systémom operačného systému. Glib poskytuje rôzne dátové štruktúry a funkcie uľahčujúce prácu s nízkoúrovňovými časťami jazyka C ako napr. práca s vláknami či mutexami. Obe knižnice sú závislosťami vykresľovacieho jadra WebKitGTK, a tak ich použitie v blokovacom jadre nezhoršuje prenositeľnosť rozšírenia.

Celá blokovacia logika je uložená v hlavičkovom súbore `adblock.h`. Zahrnutie tohto súboru do programu umožňuje prístup k funkcii `is_ad`, ktorá na základe predaných argumentov funkcie vracia hodnotu pravdivostného typu. Táto hodnota volajúceho informuje o tom, či má byť požiadavka zablokovaná (hodnota `true`), alebo prepustená ďalej (hodnota `false`).

Inicializácia štruktúr a spracovanie filtrovacích zoznamov

Pred prvým použitím funkcie `is_ad` je najprv nutné inicializovať všetky dátové štruktúry potrebné pri vyhodnocovaní zasielaných požiadaviek pomocou funkcie `adblock_init`. Tá

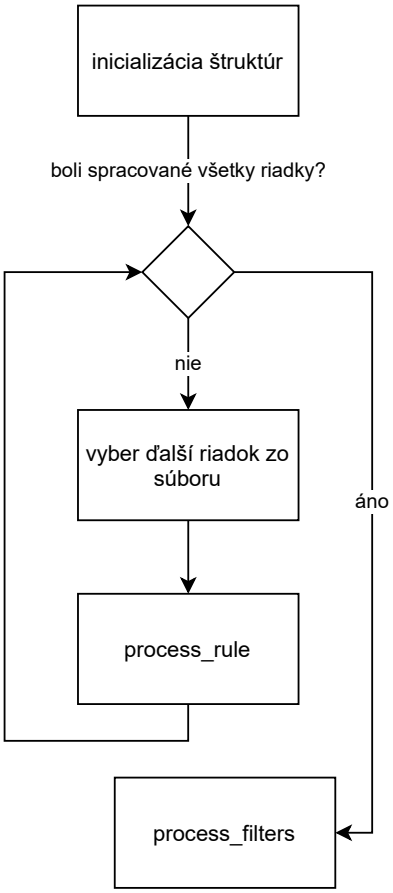
[1] https://developer.gnome.org/glib/stable/
[2] https://developer.gnome.org/gio/stable/

zavolá funkcie ako napr. `g_hash_table_new_full`, ktoré umožňujú vytvoriť a nakonfigurovať základné vlastnosti konkrétnych dátových štruktúr, pričom vracajú ukazovateľ na miesto v pamäti, kde sa novovytvorené štruktúry nachádzajú. Bez tohto kroku by so štruktúrami nebolo možné ďalej pracovať.

Po inicializovaní potrebných dátových štruktúr je ešte nutné spracovať všetky aktívne filtrovacie zoznamy. Každý riadok zoznamov prechádza filtrovacou funkciou `process_rule`. Tá odstráni prázdne riadky, riadky označené ako komentáre, ale taktiež riadky predstavujúce kozmetické a regexové filtrovacie pravidlá, ktoré v tejto verzii blokovača ešte neboli podporované. Zostávajúce riadky boli následne rozdelené do 2 skupín: výnimkové a blokovacie filtre (bližšie popísané v sekcii 2.1). Rozdelenie prebieha pridaním riadku do príslušného Glib poľa na základe toho, či pravidlo začína prefixom `@@`, ktorý označuje výnimkové pravidlá.

Po rozdelení všetkých riadkov je nad oboma poľami zavolaná funkcia `process_filters`. Tá jednotlivé riadky spracuje a uloží do finálnej dátovej štruktúry – hašovacej tabuľky, ktorá je už použitá pri samotnom vyhodnocovaní požiadaviek v blokovacej logike. Pred spracovaním pravidiel je najprv nutné vyhodnotiť histogram vyjadrujúci počet výskytov tokenov obsiahnutých v pravidlách. Ideálnou dátovou štruktúrou je v tomto prípade ďalšia hašovacia tabuľka, kde kľúč predstavuje token a jeho hodnota počet výskytov vo všetkých pravidlách.

UML diagram inicializácie dátových štruktúr a spracovania filtrovacích zoznamov pomocou funkcie `adblock_init` je dostupný vo forme obrázku 5.1.



Obr. 5.1: UML diagram ilustrujúci vykonanie funkcie `adblock_init`.

Tokeny sú z pravidla získané pomocou funkcie `tokenize_rule`, ktorá vracia ukazovateľ na novovytvorené pole pozostávajúce zo všetkých tokenov obsiahnutých v pravidle. Na analýzu a rozdelenie pravidla sú použité viaceré funkcie:

- `strchr` – Vracia ukazovateľ na prvý výskyt znaku v reťazci.
- `strstr` – Vracia ukazovateľ na prvý výskyt podreťazca v reťazci.
- `strtok` – Rozdeľuje reťazec na tokeny, ktoré sú od seba oddelené množinou separátorov.
- `g_strrstr` – Vracia ukazovateľ na posledný výskyt podreťazca v reťazci.

V prípade, že pravidlo obsahuje možnosť (vysvetlené v podsekcii 2.1) špecifikujúcu zoznam doménových mien, na ktoré sa pravidlo vzťahuje, budú tieto domény taktiež spracované a zaradené medzi tokeny.

Po vypočítaní histogramu tokenov začne program opäť prechádzať každým pravidlom v zozname. Pravidlo je znova rozdelené na tokeny a pomocou histogramu je vybraný token s najnižším výskytom vo všetkých pravidlách. Tento token bude vo finálnej hašovacej tabuľke odkazovať na pole obsahujúce toto a ďalšie pravidlá identifikované týmto tokenom.

Pravidlá v poli nebudú reprezentované reťazcom, ale špeciálnou štruktúrou `Rule`, ktorej deklarácia je uvedená vo výpise 5.1.

```
struct Rule
{
    char *pattern;
    char options;
    char used_options;
    GArray *domains;
    GArray *neg_domains;
};
```

Výpis 5.1: Deklarácia štruktúry `Rule` slúžiacej na internú reprezentáciu filtrovacích pravidiel.

Vysvetlenie jednotlivých premenných tejto štruktúry:

- `pattern` – Reťazec odkazujúci na hlavnú časť pravidla bez prefixu určujúceho typ pravidla.
- `options` – Bitové pole určujúce hodnoty použitých možností pravidla.
- `used_options` – Bitové pole určujúce, ktoré hodnoty pravidla boli použité.
- `domains` – Pole domén, na ktoré sa pravidlo vzťahuje (možnosť `$domain`).
- `neg_domains` – Pole domén, na ktoré sa pravidlo nevzťahuje (použitie kľúčového znaku negácie `~` v možnosti `$domain`).

Dôvodom na použitie štruktúry obsahujúcej všetky potrebné informácie o už spracovanom pravidle namiesto pôvodného reťazca pravidla, bola snaha o zjednodušenie, ale najmä zrýchlenie blokovacej logiky. Pravidlá sú vďaka tomuto prístupu spracované len raz pri spustení programu a nie je nutné ich analyzovať pri každej novej požiadavke.

Reprezentovanie možností bitovým poľom

K zvýšeniu rýchlosti vyhodnocovania blokovacej logiky tiež prispieva použitie bitových polí na reprezentáciu použitých možností. Prvá verzia môjho blokovacieho jadra podporovala len 8 filtrovacích možností. Z toho dôvodu bolo najvhodnejším riešením bitové pole reprezentovať dátovým typom `char`, ktorý má v jazyku C veľkosť 1 bajt. Je teda schopné uchovať až 8 pravdivostných hodnôt. V prípade rozšírenia podporovaných možností by bolo možné použiť dátový typ `short`, ktorého veľkosť dosahuje 2 bajty. Logika za použitím bitových polí na reprezentáciu možností pravidla spočíva v ich binárnej povahe – hodnoty všetkých možností je možné vyjadriť pravdivostnou hodnotou `true` alebo `false`, okrem typu `$domain`, ktorý nadobúda hodnoty poľa. Pre väčšinu možností musí byť premenná uchovávajúca jej hodnotu schopná vyjadriť nasledujúce 3 stavy:

- V pravidle sa možnosť nenachádza.
- V pravidle sa možnosť nachádza (napr. `$script`).
- V pravidle sa nachádza negovaná možnosť (napr. `$~script`).

Na uchovanie hodnoty možnosti teda postačí 1 bit, pričom je potrebný ešte 1 bit na signalizovanie faktu, že možnosť bola použitá a hodnota v prvom bite má byť braná do úvahy. Z toho dôvodu sú potrebné 2 bitové polia. Vďaka tomuto prístupu sú informácie o možnostiach uložené kompaktnejšie ako pri použití pôvodných reťazcov, no najväčšou výhodou je zjednodušenie kódu a zvýšenie efektivity. V rámci demonštrácie bolo vybrané blokovacie pravidlo uvedené vo výpise 5.2.

```
||camclips.cc/api/$image,script
```

Výpis 5.2: Filtrovacie pravidlo prevzaté zo zoznamu EasyList.

Typy možností sú v kóde reprezentované enumeračným dátovým typom (angl. enumerated type), ktorého deklarácia je uvedená vo výpise 5.3.

```
enum options
{
    Image,
    Font,
    Css,
    Popup,
    Other,
    Script,
    Object,
    Document
};
```

Výpis 5.3: Deklarácia enumeračného typu reprezentujúceho typy možností.

Bitové pole `options` je inicializované na hodnotu 0. Pomocou ľavého bitového posunu a bitového súčtu (OR) sú bity poľa na indexoch vyjadrených hodnotami `Image` a `Script` nastavené na hodnotu 1. Výpis 5.4 ukazuje kód použitý na vykonanie tejto operácie.

```
    char options = 0;
    options |= 1 << Image; // Image = 0
    options |= 1 << Script; // Script = 5
```

Výpis 5.4: Nastavenie hodnôt aktívnych možností bitového poľa podľa dodaného pravidla.

Ľavý bitový posun `1 << n` posúva hodnotu 1, binárne 0000 0001, o `n` miest doľava. Bitovým súčtom sa bit na indexe `n` nastaví na hodnotu 1, pričom ostatné indexy zostávajú bezo zmeny. Výsledná hodnota bitového poľa `options` sa po vykonaní kódu uvedeného vyššie rovná 17, binárne 0001 0001.

Po nastavení bitového poľa vyjadrujúce hodnoty jednotlivých možností je ešte nutné nastaviť premennú `used_options`, ktorá uchováva informáciu o tom, či sú jednotlivé možnosti aktívne. Toto nastavenie je vyjadrené výpisom 5.5.

```
    char used_options = 0;
    used_options |= 1 << Image; // Image = 0
    used_options |= 1 << Script; // Script = 5
```

Výpis 5.5: Nastavenie aktívnych možností bitového poľa podľa dodaného pravidla.

Keďže je výsledná hodnota rovnaká ako hodnota premennej `options`, môže tento krok na prvý pohľad pôsobiť redundantne. Jeho nevyhnutnosť je odôvodnená existenciou kľúčového znaku negácie `~`. Jeho použitie je v bitovom poli `options` reprezentované hodnotou 0. Bez ďalšieho bitového poľa by teda nebolo možné vyjadriť stav, v ktorom sa možnosť v pravidle nenachádza ani v obyčajnom a ani v negovanom stave. Pri použití negovanej možnosti, napr. `~font`, je teda nutné bitu poľa `options`, ktorý sa nachádza na indexe `Font`, priradiť hodnotu 0 (resp. tento krok úplne vynechať) a v poli `used_options`, priradiť bitu na rovnakom indexe hodnotu 1 (uvedené vo výpise 5.6).

```
used_options |= 1 << Font; // Font = 1
```

Výpis 5.6: Nastavenie hodnoty na indexe aktívnej možnosti bitového poľa použitých možností.

Pri prijatí novej požiadavky na vyhodnotenie budú z jej URL adresy, rovnakým spôsobom ako pri spracovaní pravidla, extrahované 2 bitové polia popisujúce jej možnosti. Následné rozhodnutie o tom, či sa podľa použitých možností pravidlo vzťahuje na prijatú požiadavku, je triviálne. Porovnanie a následné prerušenie vyhodnocovania je uvedené vo výpise 5.7.

```
if ((req_options & rule.used_options) != rule.options)
    continue;
```

Výpis 5.7: Porovnanie možností pravidla a možností testovanej požiadavky.

Bitové pole `req_options` určuje, ktoré možnosti definujú požiadavku. V prípade, že požiadavka je typom jednej z možností, je adekvátny bit nastavený na hodnotu 1. Napr. ak požiadavka žiada o súbor typu obrázok (napr. má koncovku `.png`), bude bit na indexe `Image` nastavený na 1. Hodnota 0, na rozdiel od bitového poľa pravidla, nereprezentuje negáciu možnosti, ale fakt, že požiadavka nie je tohto typu možnosti. Možnosť požiadavky nemôže byť negovaná, keďže požiadavka typu všetkých typov okrem jedného nedáva zmysel.

Výstupom aplikovania bitového súčinu (AND) s premennou `rule.used_options` indikujúcou, ktoré možnosti pravidla sú aktívne, je bitové pole obsahujúce hodnoty len tých možností požiadavky, ktoré sú pre pravidlo dôležité. Následným porovnaním tejto hodnoty s premennou `rule.options` je možné rýchlo zistiť, či sa pravidlo vzťahuje na požiadavku a v prípade, že nie, vyhodnocovanie predčasne ukončiť príkazom `continue`.

Po spracovaní všetkých dostupných pravidiel do 2 hašovacích tabuliek – jedna určená pre blokovacie a druhá pre výnimkové pravidlá – je blokovacie jadro pripravené na vyhodnocovanie požiadaviek.

Vyhodnocovanie požiadaviek

Funkcii `is_ad` je pomocou argumentov predaná URL adresa požiadavky, URL adresa zdroja požiadavky a typ požiadavky (bližšie popísané v sekcii 5.3). Na základe typu požiadavky je najprv vytvorené bitové pole možností. URL adresa požiadavky je pomocou špeciálnej verzie tokenizačnej funkcie upravenej na rozdeľovanie URL adries, rozdelená na tokeny. Program postupne prechádza jednotlivými tokenmi a snaží sa ich nájsť v mapovacej hašovacej tabuľke. V prípade zhody je z tabuľky získaný ukazovateľ na pole všetkých pravidiel zaradených pod týmto tokenom.

Program začne prechádzať jednotlivými pravidlami poľa. Bitové pole možností pravidla je najprv porovnané s bitovým poľom požiadavky. Ak algoritmus vyhodnocovanie predčasne neukončí, presúva sa na kontrolu doménového mena požiadavky. Prechádza poľom doménových mien, na ktoré sa pravidlo vzťahuje a hľadá zhodu s adresou požiadavky. Rovnaká kontrola prebehne nad negovanými doménovými menami pravidla. Po vyhovení všetkým testom sa požiadavka dostáva k poslednému testu na kontrolu zhody URL adresy požiadavky so vzorom pravidla, uloženého v premennej `pattern`, pomocou funkcie `match_rule`. Táto funkcia najprv URL adresu očistí odstránením všetkých nepotrebných častí ako je napr. protokol (`http/https`) alebo prefix `www`. Vzor pravidla už bol očistený (od zbytočných prefixov indikujúcich typ pravidla alebo reťazca zoznamu možností) pri hromadnom spracovaní pravidiel pri spustení programu. Vzor bol okrem toho ešte upravený pomocou špeciálnej syntaxe potrebnej na vyhodnotenie porovnania s URL adresy.

Po očistení sú oba reťazce predané funkcii `match_pattern`, ktorej návratová hodnota je vrátená funkciou `is_ad`. Funkcia `match_pattern` slúži na zistenie, či testovaný reťazec zodpovedá danému vzoru. Podporovaná syntax vzoru disponuje len 2 kľúčovými znakmi:

- `*` – Reprezentuje ľubovoľný (aj nulový) počet akýchkoľvek znakov.
- `^` – Reprezentuje akýkoľvek ne-alfanumerický znak (vrátane konca riadku) okrem znakov `_`, `-`, `.` a `%`.

Funkcia `match_pattern` je implementovaná rekurzívnym volaním samej seba. Pri každom volaní je skontrolované, či aktuálny znak vstupného reťazca vyhovuje množine prípustných znakov požadovaných vzorom. Pri zhode je porovnávaný reťazec posunutý o jeden znak doprava a funkcia je zavolaná opäť. Špeciálny prípad predstavuje výskyt znaku `*`, ktorý je popísaný vo výpise 5.8.

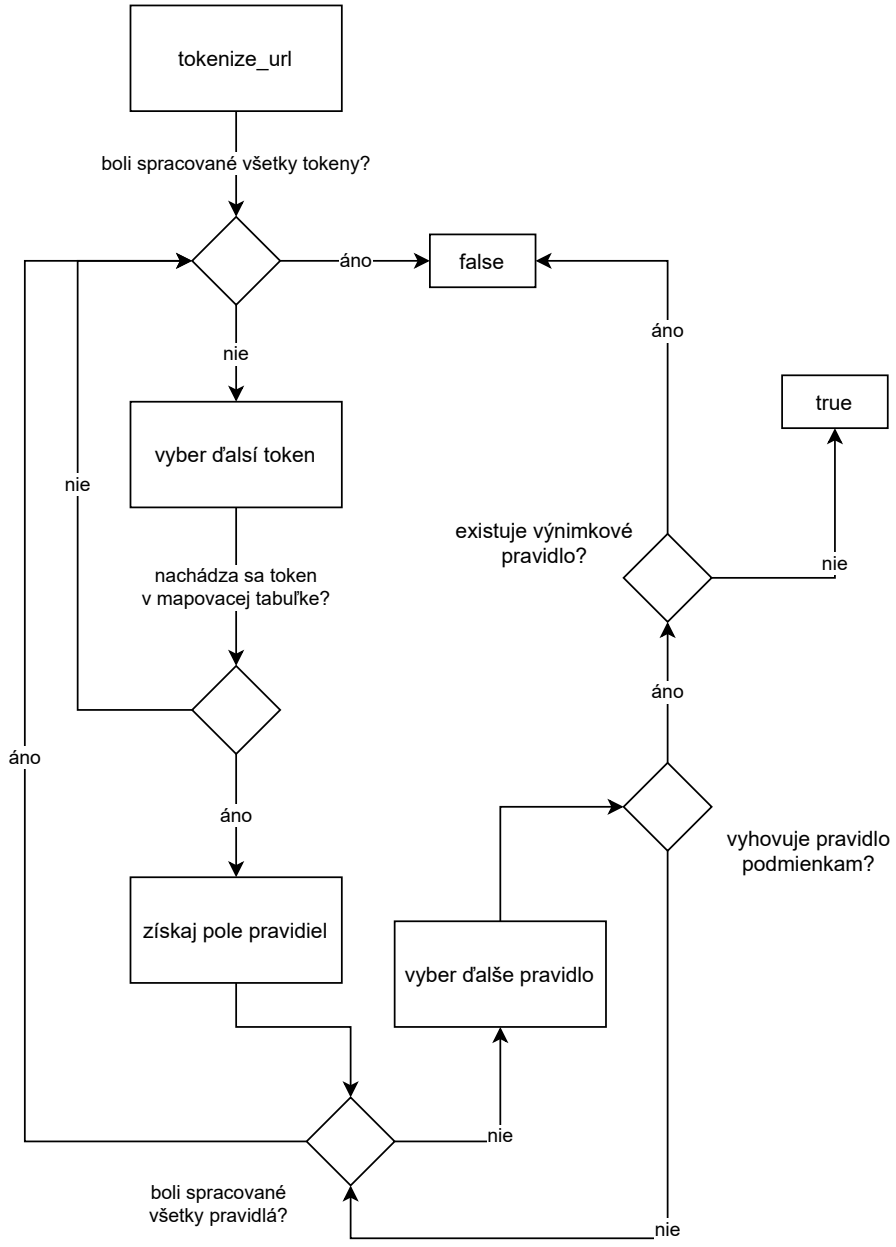
```
if (pat[0] == '*')
{
    if (str[0] == '\0' || pat[1] == '\0')
        return TRUE;

    for (int i = 0; i < strlen(str); i++)
        if (match_pattern(str + i, pat + 1))
            return TRUE;
}
```

Výpis 5.8: Ukážka kódu reagujúceho na nález špeciálneho znaku hviezdničky `*` v testovanom reťazci.

Premenná `pat` reprezentuje vzor pravidla a `str` zasa testovaný reťazec. V prípade, že sú oba reťazce na konci, funkcia vracia hodnotu `true`, ktorá sa spätne prejaví v prvom volaní funkcie `match_pattern`. V prípade, že reťazce ešte nie sú na konci, je pre správne vyhodnotenie nutné zavolať funkciu `match_pattern` pre každý zostávajúci znak testovaného reťazca. Toto rozvetvenie je nutné pre otestovanie každej potenciálnej možnosti.

V prípade, že URL adresa vyhovuje vzoru pravidla, je celé testovanie požiadavky vrátane porovnávania možností zopakované ešte raz nad zoznamom výnimkových pravidiel. Až po nenájdení zhody v druhom kole testovania je požiadavka definitívne označená za nežiadúcu.

UML diagram určený na vizualizáciu funkcie `is_ad` je dostupný vo forme obrázku 5.2.



Obr. 5.2: UML diagram ilustrujúci vykonanie funkcie `is_ad`. Kontrola možností, porovnanie domén a porovnanie výnimkových domén je reprezentované vetvením „vyhovuje pravidlo podmienkam?“

5.2 Blokovacie jadro v jazyku Rust

V pokročilej fáze vývoja sa počas testovania objavil problém s využitím prostriedkov operačného systému. WebKitGTK pre každé nové okno, resp. kartu prehliadača, vytvára nový webový proces. Každý takýto proces po vytvorení načíta a spustí všetky pridelené rozšírenia. To pri bežných rozšíreniach nepredstavuje väčší problém, no pri rozšírení určenom na blokovanie požiadaviek a nežiadúcich prvkov môže viesť k citeľnej strate výkonu, ale najmä

operačnej pamäte. Rozšírenie musí po spustení spracovať a načítať filtrovacie zoznamy, ktoré u bežných užívateľov obsahujú viac ako 100 000 pravidiel (podľa tabuľky 2.1).

Vhodným riešením tohto problému je určitá forma medzi-procesovej komunikácie na zasielanie dát do aktívnych rozšírení. Ideálnou sa javí architektúra klient-server, pri ktorej sa jednotlivý klienti – aktívne rozšírenia – pripájajú k centrálnemu serveru ako zdroju blokovacej logiky. Na serveri budú uložené všetky potrebné dátové štruktúry a rozšírenia budú slúžiť len ako určitá forma tenkého klienta.

Od komunikácie bude vyžadované súčasné pripojenie viacerých klientov a ich obojsmerná komunikácia so serverom. Vo výsledku bola zvolená technológia unixových soketov, ktorá na rozdiel od alternatívnych spôsobov medzi-procesovej komunikácie umožňuje obojsmerné spojenie bez nutnosti vytvárania vlastných mechanizmov. Ako už z názvu vyplýva, unixové sokety nie sú podporované platformou Windows, čo ale nepredstavuje problém, keďže samotná technológia WebKitGTK na nej nie je natívne podporovaná.

Využitie knižnice adblock-rust

Použitie medzi-procesovej komunikácie okrem šetrenia prostriedkov prináša taktiež voľnosť vo výbere použitého implementačného programovacieho jazyka. Na rozdiel od vyššie spomínaného language binding-u neprináša žiadne nevýhody, keďže podpora unixových soketov je súčasťou štandardnej knižnice väčšiny programovacích jazykov. Jedným z takých jazykov je i Rust, ktorý bol použitý na vytvorenie knižnice určenej na blokovanie požiadaviek pre prehliadač Brave. Rozhodol som sa pre zanechanie rozpracovaného blokovacieho jadra a jeho nahradenie serverom napísaným v jazyku Rust, s použitím knižnice adblock-rust. Knižnica je vyvíjaná pre účely komerčného prehliadača, ktorý mesačne využíva viac ako 25 miliónov užívateľov. Je teda adekvátne očakávať jej vysokú kvalitu a taktiež rýchle a dlhodobé aktualizácie reagujúce na zmeny v stále meniacej sa oblasti reklám a sledovacích prvkov. Takýchto kvalít, by sa mne, ako jednotlivcovi, dosahovalo pomerne ťažko.

Výhodou programovacieho jazyka Rust oproti jazyku C je, že jeho prekladač je navrhnutý tak, aby pre každý úspešne preložený program garantoval pamäťovú bezpečnosť (angl. memory safety), čo sa o jazyku C tvrdiť nedá. Výnimku predstavuje použitie kľúčového slova `unsafe`. Všetok kód v takto označenom bloku žiadne garancie neposkytuje. V mojom riešení ale túto konštrukciu nepoužívam.

Server s blokovacou logikou

Filtrovací server musí vykonávať 2 hlavné úlohy: správa filtrovacích zoznamov a vyhodnocovanie požiadaviek na filtrovanie. Správa zoznamov zahŕňa automatické aktualizovanie aktívnych filtrovacích zoznamov a možnosť ich konfigurácie.

Po spustení serveru je okamžite zavolaná funkcia `setup_blocker`, ktorá vytvorí všetky potrebné zložky a súbory (v prípade, že ešte neexistujú):

- Priečinok `lists` – Slúži na ukladanie filtrovacích zoznamov.
- Súbor `urls` – Konfiguračný súbor na správu filtrovacích zoznamov.
- Súbor `custom` – Obsahuje vlastné filtrovacie pravidlá vytvorené užívateľom. Nachádza sa v zložke `lists`.
- Centrálny priečinok `ars` – Sú v nej uložené všetky vyššie spomínané súbory a zložky.

Priečinok `ars` sa nachádza v konfiguračnej zložke užívateľa – `/home/<user>/.config/ars`, kde reťazec `<user>` predstavuje užívateľské meno. Cesta k domácemu adresáru je získaná pomocou prečítania z premennej prostredia `HOME`.

Súbor `urls` obsahuje zoznam aktívnych filtrovacích zoznamov. Každý riadok súboru predstavuje jeden filtrovací zoznam. Tie sú ukladané vo formáte uvedenom vo výpise 5.9.

```
<URL adresa filtrovacieho zoznamu> <casova znacka>
```

Výpis 5.9: Formát filtrovacích zoznamov v konfiguračnom súbore `urls`.

Použitá URL musí priamo odkazovať na požadovaný filtrovací zoznam. Časová značka reprezentuje moment v unixovom čase, kedy má byť daný zoznam aktualizovaný. Kontrola expirácie filtrovacích zoznamov a prípadné aktualizovanie vybraných zoznamov, prebieha vždy po spustení serveru. Čas expirácie je zo zoznamov získaný z položky `! Expires: n days`, kde `n` značí interval aktualizácií v dňoch. Táto hodnota sa pripočíta k aktuálnemu unixovému času a pripojí k URL adrese zoznamu. Pri pridávaní nového filtrovacieho zoznamu do tohto súboru je časovú značku možné vynechať s tým, že bude vygenerovaná pri najbližšej aktualizácii zoznamu.

Štandardná knižnica jazyka Rust nedisponuje žiadnym HTTP klientom, ktorý je potrebný na aktualizáciu filtrovacích zoznamov. Je teda nutné použiť knižnicu vyvíjanú treťou stranou. HTTP klient bude slúžiť len na sťahovanie filtrovacích zoznamov, ktoré sú distribuované formou plain text súborov. Najpoužívanejšie knižnice tohto typu sú pre potreby práce zbytočne komplexné, čo so sebou potenciálne prináša zvýšený výskyt bezpečnostných chýb. Na základe diskusie na oficiálnom fóre jazyka Rust[3] a nezávislého článku o testovaní jeho HTTP klientov[4], som zvolil knižnicu `attohttpc`. Tá splňuje moje požiadavky na malú komplexnosť a jednoduchosť použitia. Stiahnutie súboru zo vzdialeného serveru je v nej možné vykonať jedným riadkom kódu, ktorý je uvedený vo výpise 5.10.

```
let res = attohttpc::get(&url).send().unwrap();
```

Výpis 5.10: Zaslanie požiadavky GET na špecifikovanú adresu.

Po spracovaní a prípadnej aktualizácii filtrovacích zoznamov je v centrálnej zložke vytvorený súbor `engine`. Aktualizované blokovacie jadro je serializované do binárneho formátu a uložené do tohto súboru. Ak od poslednej serializácie jadra nedošlo k žiadnym zmenám v konfiguračnom súbore filtrovacích zoznamov, a ani v súbore užívateľom vytvorených pravidiel, a zároveň nie je nutná aktualizácia žiadnych filtrovacích zoznamov, je pri ďalšom spustení serveru jadro načítané a deserializované z tohto súboru. Server tak nemusí zbytočne spracovávať zoznamy pri každom spustení, vďaka čomu je inicializácia jadra efektívnejšia a rýchlejšia.

Funkcia `setup_blocker` disponuje argumentom enumeračného dátového typu, ktorým možno určiť typ inicializácie filtrovacieho jadra:

- `Default` – Bežná inicializácia.
- `Reload` – Vynucuje "reštartovanie"jadra, vďaka čomu sa prejavia zmeny vykonané v konfiguračných súboroch.

[3] https://users.rust-lang.org/t/lightweight-alternative-for-reqwest/33601
[4] https://medium.com/@shnatsel/smoke-testing-rust-http-clients-b8f2ee5db4e6

- `Update` – Vynucuje aktualizáciu všetkých filtrovacích zoznamov, i keď to podľa časových známok nie je nutné. Okrem toho má rovnaký efekt ako typ `Reload`.

Pri spustení serveru je jadro vždy inicializované s typom `Default`. Zvyšné typy sú používané pri interakcii užívateľa so serverom pomocou užívateľského rozhrania, viď kapitola o GUI 5.4.

Komunikácia so serverom

Komunikácia medzi serverom a jednotlivými klientmi prebieha pomocou unixových soketov s využitím štruktúr modulu `std::os::unix::net`, ktorý je súčasťou štandardnej knižnice jazyka Rust. Demonštrácia spustenia serveru je uvedená vo výpise 5.11.

```
let listener = UnixListener::bind(socket_path);
println!("init-done");

let blocker = Arc::new(blocker);

for stream in listener.incoming() {
    match stream {
        Ok(stream) => {
            let blocker = blocker.clone();
            thread::spawn(move || handle_client(stream, blocker));
        }
        ...
    }
}
```

Výpis 5.11: (Ukážka popisujúca spustenie serveru a následné vytváranie vlákien pre každého klienta. Vynecháva spracovanie výnimiek za účelom skrátenia kódu.

Pomocou funkcie `UnixListener::bind` začne server načúvať na sockete s cestou určenou argumentom tejto funckie. Správa `init-done` je vypísaná na štandardný výstup za účelom informovania klienta, ktorý sa pokúšal server spustiť, o úspešnej inicializácii blokovacieho jadra a o možnosti začať odosielať požiadavky na server. Premenná `blocker` obsahujúca štruktúru už inicializovaného blokovacieho jadra je nahradená ukazovateľom typu `Arc`, odkazujúcim na pôvodnú štruktúru blokovacie jadra `blocker`. Tento druh ukazovateľa umožňuje bezpečné zdieľanie hodnoty, ktorú uchováva, medzi vláknami. Funkcia `incoming` vracia iterátor prichádzajúcich spojení, ktorými sa následne začne prechádzať. V prípade, že je vrátené spojenie v poriadku, je vytvorená kópia premennej `blocker`. Jej vlastníctvo (angl. ownership) je následne predané novovytvorenému vláknu pomocou kľúčového slova `move`. Vlákno spúšťa funkciu `handle_client`, ktorá má na starosti komunikáciu s klientom. Použitie vlákien umožňuje obsluhu viacerých klientov súčasne, čo je výhodné napríklad pri otvorení viacerých kariet/okien prehliadača WebKitGTK.

Knižnicu `adblock-rust`, ale nie je možné pri predvolených nastaveniach zdielať medzi vláknami. V pôvodnej verzii to možné bolo, no pod zámienkou zvýšenia efektivity a rýchlosti vyhodnocovania knižnice, bolo použitie ukazovateľa typu `Arc` nahradené ukazovateľom typu `Rc`. Ten taktiež umožňuje zdieľanie hodnoty, ktorú uchováva, medzi viacerými vlastníkmi, no na rozdiel od prvého spomínaného typu je určený na použitie v kontexte len 1 vlákna.

Následkom toho je, že Rust vyššie uvedený kód vôbec nezkompiluje. Po viacerých prosbách od užívateľov o navrátenie sa k pôvodnému prístupu s využitím typu `Arc` bol vytvorený tzv. „pull request“, ktorý bol neskôr začlenený do kódu knižnice[5]. Ten stále používal ukazovateľ `Rc`, no zmenou konfiguračného súboru knižnice umožňoval prepnutie na typ `Arc`.

Konfiguračný súbor všetkých Rust „balíčkov“, `Cargo.toml`, disponuje sekciou `[features]`. V tejto sekcii je možné ľubovoľné prepínanie medzi vybranými funkciami programu, v tomto prípade vyššie spomínanej zmeny použitého typu ukazovateľa dostupnej pod názvom `object-pooling`. Upravená sekcia konfiguračného súboru je uvedená vo výpise 5.12.

```
[dependencies.adblock]
version = "0.3.10"
default-features = false
features = ["embedded-domain-resolver", "full-regex-handling"]
```

Výpis 5.12: Časť konfiguračného súboru `Cargo.toml` upravená za účelom vypnutia funkcie `object-pooling`.

Jednotlivé funkcie nie je možné granulárne deaktivovať a z toho dôvodu je nutné vypnutie všetkých predvolených funkcií a následné vybratie všetkých požadovaných funkcií.

Funkcia `handle_client` postupne spracúva každý riadok správ odoslaných klientom. Riadok rozdelí na časti oddelené medzerami, pričom prvá časť pozostávajúca vždy z 1 znaku, indikuje o aký typ požiadavky ide. Úsek kódu, ktorý rozdeľuje požiadavky podľa typu s použitím kľúčového slova `match`, je uvedený vo výpise 5.13.

```
match req_type {
    "n" => {
        // network request
    },
    "c" => {
        // cosmetic request
    },
    "r" => {
        // reload engine request
    },
    "u" => {
        // force update request
        },
    _ => {
        // unknown code supplied
        }
    };

    stream.write(res.as_bytes()).unwrap();
}
```

Výpis 5.13: Rozdelenie požiadaviek podľa typu.

Server tak spolu podporuje 4 typy požiadaviek:

[5] https://github.com/brave/adblock-rust/issues/106

- **Sieťová požiadavka** – formát `n <request_url> <source_url> <request_type>` – Na základe argumentov vyhodnocuje nežiadúcnosť požiadavky. Vracia znak `1`, ak je požiadavka nežiadúca, v opačnom prípade vracia znak `0`.
- **Kozmetická požiadavka** – formát `c <website_url> <ids> <classes>` – Argumenty predstavujú URL adresu stránky, zoznam identifikátorov a názvov tried prvkov, ktoré sa nachádzajú na stránke. Odpoveďou je rozsiahle CSS pravidlo, ktoré po vložení do stránky skryje všetky nežiadúce prvky.
- **Požiadavka na znovu-načítanie filtrovacieho jadra** – formát `r` – Bez argumentov, odpoveďou je znak `0`, ktorý indikuje úspešne spustenie jadra.
- **Požiadavka na vynútenú aktualizáciu aktívnych filtrovacích zoznamov** – formát `u` – Bez argumentov, odpoveďou je znak `0`, ktorý indikuje úspešne spustenie jadra.

Po spracovaní požiadavky a vykonaní požadovanej akcie, je odpoveď uložená do premennej `res` a pomocou funkcie `write` odoslaná späť klientovi, ktorý ju požadoval.

Požiadavky kladené na interakciu s filtrovacím jadrom sú detailnejšie popísané v kapitole o GUI 5.4.

5.3 Rozšírenie pre WebKitGTK

V prípade tejto práce rozšírenie predstavuje len určitú formu prepojenia prehliadača a blokovacej logiky. Rozšírenie slúži na získanie informácií o jednotlivých požiadavkách, ktoré sú potrebné na vyhodnotenie výsledkov blokovania. Tie sú následne predané blokovacej logike, ktorá rozšíreniu vracia odpoveď a to na jej základe vykoná potrebnú operáciu. Úloha rozšírenia teda spočíva v získavaní informácií o požiadavkách/prvkoch stránky a schopnosti správne zareagovať na podnet od blokovacej logiky, k zablokovaniu požiadavky alebo odstránenia prvku stránky.

Získavanie informácií

Požiadavky

Blokovacia logika potrebuje na vyhodnotenie požiadavky 3 údaje:

- URL adresa stránky, z ktorej bola požiadavka odoslaná – napr. http://example.com/test
- URL adresa požiadavky – napr. http://example.com/test/image.png
- Typ požiadavky – napr. image

Obe URL adresy je možné získať pomocou funkcií popísaných v sekcií 3.3.

Typ požiadavky nadobúda hodnotu jedného z typov definovaných WebExtensions štandardom. Podľa dokumentácie k štandardu ide o reťazec reprezentujúci kontext, v ktorom bolo požiadané o zdroj v požiadavke [7]. Typ požiadavky teda môže predstavovať napr. typ dát, o ktoré v požiadavke žiadame (napr. „font“), ale aj spôsob akým o tieto dáta žiadame (napr. „xmlhttprequest“). Podľa zdrojového kódu knižnice adblock-rust je ale možné vidieť, že využíva len časť z dostupných typov[6], ktoré zoskupuje do enumeračného dátového typu s hodnotami:

[6] https://github.com/brave/adblock-rust/blob/94b377ca106dda88d301d6089639271daea1117a/src/request.rs#L46

- **Ping** – Zahŕňa typy „beacon“ (odoslané pomocou Beacon API) a „ping“ (atribút ping pri hyperlinku).
- **Csp** – Typ „csp_report“ – požiadavky odoslané pri porušení Content-Security-Policy.
- **Document** – Typ „document“.
- **Font** – Typ „font“.
- **Image** – Typ „image“.
- **Media** – Typ „media“ – zdroje načítane pomocou `<video>` alebo `<audio>` prvkov.
- **Object** – Typ „object“ – zdroje načítané pomocou `<object>` alebo `<embed>` prvkov.
- **Script** – Typ „script“.
- **Stylesheet** – Typ „stylesheet“.
- **Subdocument** – Typ „subdocument“ – zdroje načítané pomocou `<iframe>` alebo `<frame>` prvkov.
- **Websocket** – Typ „websocket“.
- **Xmlhttprequest** – Typ „xmlhttprequest“.
- **Other** – Typ „other“.

WebKitGTK API žiadnym mechanizmom na získanie typu požiadavky nedisponuje. Najviac sa k požadovanej fukncionalite približuje funkcia `webkit_uri_request_get_http_headers`, ktorá vracia hlavičky požiadavky. Keďže pracujeme s ešte neodoslanou požiadavkou, nemáme prístup k položke `Content-Type` odpovede. Zostáva nám pracovať už len s položkou `Accept`, ktorá nemusí byť vždy prítomná a v niektorých prípadoch obsahuje niekoľko rôznych hodnôt, čo by vyústilo ku zhode s viacerými typmi.

Z toho dôvodu je nutné uchýliť sa k pomerne naivnému, no funkčnému prístupu, ktorý spočíva v jednoduchej kontrole koncoviek jednotlivých požiadaviek na zhodu s množinami sufixov. Riešenie je uvedené vo výpise 5.14.

```
if (g_str_has_suffix(path, ".js"))
    res_type = "script";
else if (g_str_has_suffix(path, ".css"))
    res_type = "stylesheet";
else if (g_str_has_suffix(path, ".jpg") || g_str_has_suffix(path, ".png"))
    res_type = "image";
...
```

Výpis 5.14: Zistenie typu požiadavky podľa koncovky jej URL.

Nevýhodou tohto prístupu je fakt, že URL adresa nemusí obsahovať koncovku indikujúcu typ požiadavky. Ďalším problémom je taktiež nemožnosť zistiť akým spôsobom bolo o dáta požiadané, resp. v akom kontexte bola požiadavka vytvorená (napr. pomocou XMLHttpRequest API).

HTML prvky stránky

Ako bolo spomenuté v sekcii 3.3, v aktuálnej verzii WebKitGTK je na manipuláciu s DOM HTML stránky nutné použiť JavaScriptCore API. Táto API umožňuje priamy prístup k Javascript jadru prehliadača. Všetku interakciu bude teda nutné vykonať v jazyku Javascript.

Na získanie zoznamu nežiadúcich prvkov je najprv nutné blokovaciemu jadru zaslať výpis `id` a `class` atribútov všetkých prvkov stránky. Tie je možné získať krátkym kódom Javascript uvedenom vo výpise 5.15.

```
// class names
Array.from(new Set([].concat.apply([], Array.from(document.
    getElementsByTagName('*')).map(elem => Array.from(elem.classList))))).
    join('\t')", -1);

// ids
Array.from(document.getElementsByTagName('*')).map(elem => elem.id).filter(
    elem => elem).join('\t')", -1);
```

Výpis 5.15: Kód v jazyku Javascript určený na získanie všetkých identifikátorov a názvov tried, ktoré sa nachádzajú v dokumente.

Oba príkazy spočívajú v prechode každým prvkom dokumentu pomocou funkcie `document.getElementsByTagName('*')` a následným premapovaním prvku na požadovaný atribút. Rozdiel je v spôsobe spracovania atribútu `class`, ktorý je ešte nutné zretaziť, keďže môže na rozdiel od atribútu `id` nadobudnúť viac ako 1 hodnotu. Nakoniec sú obe polia prekonvertované na reťazec, pomocou funkcie `join`, do formátu vyžadovaného od serveru s blokovacou logikou (jednotlivé položky oddelené znakom tabulátoru `\t`).

Zoznam hodnôt atribútov `id` a `class` je nutné získať a odoslať serveru až po načítaní všetkých požiadaviek, keďže tie môžu obsah stránky ďalej dynamicky meniť pomocou skriptov. Odpoveď, ktorú server vracia obsahuje CSS pravidlo určené na skrytie všetkých nežiadúcich prvkov stránky. Rozšírenie sa pripojí na signál `document-loaded`, ktorý signalizuje úspešné načítanie dokumentu DOM webovej stránky. Pri každom aktivovaní signálu je do stránky vložený kód v jazyku Javascript uvedený vo výpise 5.16.

```
var link = document.createElement('link');
link.rel = 'stylesheet';
link.href = 'a';
document.head.appendChild(link);
window.onload = function () {
    link.sheet.insertRule('CSS pravidlo');
}
```

Výpis 5.16: Kód na vloženie CSS pravidla do stránky.

Dôležitou časťou kódu je použitie HTML prvku typu `<link>` namiesto, na prvý pohľad vhodnejšieho, typu `<style>`. Prvok typu `<style>` často nie je možné vložiť do stránky, keďže je jeho použitie zablokované striktným nastavením CSP pravidla `style-src`. CSP (Content-Security-Policy) je hlavička odpovede na požiadavku umožňujúca kontrolovať, aké zdroje môže klient načítavať pre konkrétnu stránku [6]. Prvok `<link>` je pridaný do hlavičky dokumentu a následne je doň vložené CSS pravidlo získané od serveru. Toto pravidlo je

v kóde vyššie symbolizované reťazcom `CSS pravidlo`. Obsluha udalosti `window.onload` sa zavolá po aktivovaní udalosti `load`. Tá nastane po načítaní všetkých potrebných zdrojov, vrátane kaskádových štýlov a skriptov. Vďaka tomu sú pri filtrovaní skontrolované naozaj všetky zdroje a prvky, ktoré stránka využíva, čo zvyšuje presnosť filtrovania.

Manuálne odstránenie prvkov stránky

Rozšírenie po vzore uBlock Origin užívateľom taktiež umožňuje dočasné alebo permanentné odstránenie nežiadúcich prvkov stránky, ktoré blokovacie jadro, resp. filtrovacie zoznamy z nejakého dôvodu nezachytili. Takáto situácia môže nastať napr. keď správcovia zoznamov ešte nestihli aktualizovať svoje zoznamy na základe zmien vykonaných poskytovateľmi stránky alebo prvky nie sú všeobecne považované za nežiadúce a nevyhovujú len konkrétnemu užívateľovi.

Dočasné aj permanentné odstránenie prvkov funguje na podobnom princípe, kde sa okolo prvku vybraného myšou zobrazí farebný rámček oddeľujúci aktívny prvok od zvyšku dokumentu (zachytené na obrázku 5.3). Po ľavom kliknutí myši je prvok odstránený. V prípade permanentného odstránenia je pomocou Javascript skriptu uvedeného vo výpise 5.17 vygenerované CSS pravidlo, označujúce práve odstránený element, ktoré je následné pridané do súboru s vlastnými pravidlami, vďaka čomu bude prvok blokovaný i pri opakovaných zobrazeniach stránky.

Obr. 5.3: Ukážka módu na vyberanie prvku stránky určeného na odstránenie/zablokovanie (prehliadač vimb).

```
function sel(elem) {
    if (!elem) {
```

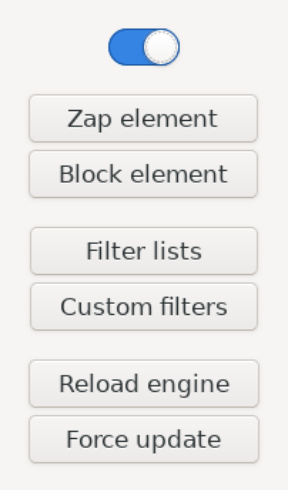
```
        return '';
    }
    if (elem.id) {
        return '#' + elem.id;
    } else {
        var index = 1;
        var el = elem.previousElementSibling;
        while (el) {
            index++;
            el = el.previousElementSibling;
        }
        return sel(elem.parentElement) + ' > ' + elem.tagName.toLowerCase()
         + ':nth-child(' + index + ')';
    }
}
```

Výpis 5.17: Vrátenie CSS selektoru daného elementu.

Funkcia určená na vrátenie CSS pravidla špecifikujúceho vybraný prvok, začína kontrolou, či prvok neobsahuje atribút `id`, ktorý ho jednoznačne identifikuje v celom dokumente. Ak nie, tak funkcia pokračuje zistením pozície prvku vo vzťahu k jeho súrodencom. Tento index spolu s triedou `:nth-child()` prvok jednoznačne určuje v aktuálnom kontexte. Táto funkcia je následne rekurzívne volaná nad rodičovským prvkom aktuálneho prvku, až kým prvok nemá nadradené prvky. Jednotlivé návratové hodnoty tejto funkcie sú zreťazené použitím tzv. „child combinator“ (symbol `>`), ktorý sa odkazuje na priamych potomkov aktuálneho prvku.

5.4 Grafické užívateľské rozhranie

Počas vyvíjania grafického užívateľského rozhrania som sa snažil o splnenie všetkých požiadaviek spomenutých v sekcii 4.2. Výsledné rozhranie je teda minimalistické a zamerané na funkčnosť. Pozostáva len z 1 okna, ktoré obsahuje 6 tlačidiel a 1 prepínač. Snímka obrazovky zachytávajúca užívateľské rozhranie je dostupná vo forme obrázku 5.4.



Obr. 5.4: Ukážka užívateľského rozhrania rozšírenia BlocKit.

Popis funkcie jednotlivých tlačidiel:

- **Prepínač** – Zapnutie resp. vypnutie blokovania požiadaviek a prvkov.
- **„Zap element“** – Dočasné manuálne odstránenie prvku stránky.
- **„Block element“** – Permanentné odstránenie prvku stránky.
- **„Filter lists“** – Otvorenie súboru obsahujúceho zoznam aktívnych filtrovacích zoznamov v predvolenom textovom editore.
- **„Custom filters“** – Otvorenie súboru obsahujúceho vlastné filtrovacie pravidlá v predvolenom textovom editore.
- **„Reload engine“** – Reštartovanie blokovacieho jadra za účelom aplikovania zmien v súbore so zoznamom aktívnych filtrovacích pravidiel alebo v súbore s vlastnými filtrovacími pravidlami.
- **„Force update“** – Vynútené aktualizovanie všetkých aktívnych filtrovacích zoznamov na najnovšiu verziu a následné rovnaké reštartovanie ako pri aktivovaní tlačidla „Reload engine“.

Každé tlačidlo disponuje nápoveďou (angl. tooltip), ktorá sa zobrazí po zotrvaní myši na danom tlačidle. Užívateľa krátkym popisom informuje o funkcii, ktorú tlačidlo vykonáva, čo uvítajú najmä nový užívatelia.

Interakcia s filtrovacími zoznamami a vlastnými filtrovacími pravidlami prebieha formou súborov, čo umožňuje užívateľom blokovacie jadro konfigurovať bez použitia grafického užívateľského rozhrania a jednoduché prepojenie s vlastnými skriptami.

Prístup k oknu s nastaveniami rozšírenia cez ikonu v pravom hornom rohu prehliadača, ktorý je bežný na alternatívnych prehliadačoch, nie je možný keďže WebKitGTK nedisponuje žiadnou možnosťou upravovať užívateľské rozhranie prehliadača z prostredia rozšírenia. Z toho dôvodu je okno s nastaveniami dostupné navigovaním prehliadača na adresu `blockit://settings`. Pôvodným zámerom bolo okno sprístupniť cez kontextovú ponuku, ktorá sa zobrazí po kliknutí pravým tlačidlom myši na akejkoľvek stránke. To ale vo výsledku

nebolo možné, kvôli nájdeniu chyby vo vykresľovacom jadre WebKitGTK, ktorá znemožňovala aktivovanie vlastných položiek pridaných do kontextovej ponuky. Chybu som sa pokúsil najprv neúspešne obísť a neskôr sám opraviť, no kvôli obmedzenému času a rozsiahlej kódovej základni WebKitGTK som chybu nakoniec len nahlásil pomocou oficiálneho kanálu[7].

Celé grafické užívateľské rozhranie je postavené na toolkit-e GTK. Ten bol zvolený z dôvodu vysokej popularity na platformách Linux a BSD. Z toolkit-u GTK taktiež pochádza vykresľovacie jadro WebKitGTK, čo z neho robí prirodzenú voľbu. Na implementáciu rozhrania bol použitý program Glade, ktorý umožňuje vizuálnu tvorbu užívateľských rozhraní nezávisle od implementačného programovacieho jazyka. Výstupom programu je schéma užívateľského rozhrania, ktorá je pomocou nástroja `glib-compile-resources` prekonvertovaná na hlavičkový súbor jazyka C. Ten je v zdrojovom súbore zahrnutý do rozšírenia, vďaka čomu nie je nutné manuálne špecifikovanie jednotlivých prvkov rozhrania a ich parametrov v kóde.

[7] https://bugs.webkit.org/show_bug.cgi?id=223160

Kapitola 6

Testovanie

Nasledujúce sekcie uvádzajú popis implementácie a výsledky viacerých testov, ktorých úlohou bolo:

- overiť, že filtrovacia logika rozšírenia BlocKit sa správa totožne s logikou použitej knižnice adblock-rust,
- poukázať na vylepšenia v oblasti filtrovania, ktoré rozšírenie BlocKit prináša oproti konkurenčnému riešeniu wyebadblock,
- overiť, že rozšírenie BlocKit spĺňa základné požiadavky kladené na filtrovacie rozšírenie pomocou nástroja tretej strany,
- zistiť, aký dopad má použitie rozšírenia BlocKit na rýchlosť prehliadania.

Posledná sekcia je venovaná komunikácii s cieľovou užívateľskou základňou a následnou spätnou väzbou od užívateľov.

6.1 Testovanie správnosti a rozsahu blokovania

Pod testovaním správnosti a rozsahu sa rozumie porovnanie reakcie rozšírenia BlocKit oproti konkurenčnému riešeniu wyebadblock a referenčnému riešeniu na veľkom množstve požiadaviek. Pre každé testované rozšírenie bude výstupom testu počet správne zablokovaných resp. nezablokovaných požiadaviek a počet tzv. false positives a false negatives – tj. koľko požiadaviek bolo nesprávne označených za nežiadúce a koľko z nich bolo naopak nesprávne označených za neutrálne.

Na tento druh testovania je potrebný rozsiahly dataset požiadaviek. Ten by sa dal získať vytvorením skriptu na prechod najnavštevovanejšími stránkami na internete a následným uloženým URL adries všetkých odoslaných požiadaviek. Problémom je získať samotný zoznam najnavštevovanejších adries. Plný prístup k nemu je často krát spoplatnený, alebo nie je možné overiť legitimitu získaných dát. Z toho dôvodu som sa rozhodol využiť voľne dostupný dataset požiadaviek[1] vytvorený spoločnosťou Cliqz. Ten spolu obsahuje 242 944 požiadaviek, ktoré boli vytvorené navštívením 500 najnavštevovanejších stránok, na ktorých boli dodatočne navštívené až 3 náhodné vybrané podstránky dostupné z domácej stránky (angl. home page). Dataset vznikol práve za účelom testovania výkonu najpoužívanejších blokovačov obsahu ako uBlock Origin či Adblock Plus [24].

[1] https://cdn.cliqz.com/adblocking/requests_top500.json.gz

Požiadavky sú v datasete reprezentované ako objekt formátu JSON s nasledujúcimi atribútmi:

- `frameUrl` – Zdrojová URL adresa, z ktorej bola požiadavka odoslaná.
- `url` – URL adresa požiadavky.
- `cpt` – Typ požiadavky (popísaný v sekcii 5.3).

Dataset neobsahuje informáciu o tom, či majú byť jednotlivé požiadavky zablokované, alebo povolené. Preto je nutné dáta najprv spracovať skriptom, ktorý pre každú požiadavku vyhodnotí reakciu referenčného riešenia a pripojí ju k zvyšným informáciám o požiadavke vo forme kľúčového slova `true` alebo `false`.

Formát datasetu nie je validný JSON. Každý riadok síce pozostáva z JSON objektu popisujúceho jednotlivé požiadavky, no medzi objektami chýba oddeľovací znak čiarky `,` a celý dokument nie je obalený hranatými zátvorkami `[` a `]` označujúce zoznam. Ide teda skôr o klasický textový súbor s prvkami JSON. Skript tak slúži aj na zmenu formátu z pseudo JSON na obyčajný textový formát, kde každý riadok obsahuje informácie o jednotlivých požiadavkách oddelené medzerou bez akéhokoľvek kľúča.

Ako referenčné riešenie bola použitá knižnica adblock-rust, ktorá je taktiež použitá v rozšírení BlocKit ako zdroj blokovacej logiky. Pre toto rozšírenie je teda adekvátne očakávať 100 % zhodu vo výsledkoch testovania s referenčným riešením. Vďaka tomu, test umožňuje odhaliť potenciálnu chybu v spôsobe, akým je knižnica použitá v rozšírení. Okrem vyvíjaného rozšírenia bude rovnakým spôsobom testované konkurenčné riešenie wyebadblock.

Z dôvodu použitia knižnice adblock-rust, bol skript implementovaný v jazyku Rust. Kvôli neštandardnému JSON formátu, homogénnym vlastnostiam objektu reprezentujúceho požiadavku (štruktúra objektu je v celom dokumente rovnaká) a absencií JSON parser-u v štandardnej knižnici jazyka Rust, boli jednotlivé riadky datasetu jednoducho spracované pomocou funkcie `split`.

Výstup skriptu je použitý ako vstup pre program napísaný v jazyku C, ktorý už vykonáva samotné testovanie. V programe sú implementované 2 funkcie, pomocou ktorých sú kontaktované servery testovaných rozšírení. Obe funkcie fungujú na rovnakom princípe, ktorý spočíva v odoslaní správy s informáciami o požiadavke na požadovaný server a následnom (tzv. blocking) čakaní na odpoveď. Rozdiel testovaných rozšírení je v použití odlišných typov medzi-procesovej komunikácie. Rozšírenie BlocKit využíva rozhranie unixových soketov, zatiaľ čo wyebadblock komunikuje pomocou pomenovaných rúr (angl. named pipe, taktiež označované ako FIFO). Hlavným rozdielom je, že pomenované rúry, na rozdiel od rozhrania unixových soketov, neumožňujú natívnu obojsmernú komunikáciu. Táto odlišnosť je ale pre koncového užívateľa nepodstatná a ide skôr o uľahčenie vývoja rozšírenia. Funkcia určená na komunikáciu s rozšírením wyebadblock bola vyňatá priamo zo zdrojového kódu tohto rozšírenia v snahe vytvoriť čo najférovejšie testovacie podmienky.

V hlavnej funkcii testovacieho programu je najprv vytvorené spojenie so serverom rozšírenia BlocKit pomocou socketu. Následne je otvorený súbor so zoznamom požiadaviek, vygenerovaný skriptom, napísanom v jazyku Rust. Program postupne prechádza každou požiadavkou, volá nad ňou funkcie na odoslanie požiadavky na server testovaného rozšírenia a výsledky porovnáva s výstupom referenčného riešenia. Podľa danej situácie sú inkrementované počítadlá vyjadrujúce počet správnych a nesprávnych odpovedí.

Testovanie bolo rozdelené na 2 časti, pričom v prvej z nich sú rozšírenia testované len na filtrovacích zoznamoch EasyList a EasyPrivacy. Druhý test už prebieha s využitím všetkých

bežných filtrovacích zoznamov, ktoré sú detailnejšie popísané v kapitole 2.2. Dôvodom je snaha poukázať na rozdiely v testovaní s menším počtom filtrovacích zoznamov. Zoznamy EasyList a EastPrivacy spoločne predstavujú určitý zlatý štandard filtrovacích zoznamov, keďže sami o sebe pokrývajú pomerne veľký počet bežných reklám a sledovacích prvkov. Vďaka zvýšeniu rýchlosti filtrovania pri použití menšieho počtu filtrovacích zoznamov je použitie len týchto 2 zoznamov pomerne rozšírené.

Na základe výsledkov testovania predstavených v tabuľke 6.1 pre rozšírenie BlocKit možno potvrdiť správnosť očakávania, že rozšírenie dosiahne rovnaké výsledky ako referenčné riešenie a dôjde k 100 % zhode. Tabuľka 6.2 referuje výsledky testovania rozšírenia wyebadblock, ktoré si v teste viedlo značne horšie. Pri testovaní s oboma zoznamami správne odhadlo len 65.5 % požiadaviek. Nesprávne odhady sú z 94 % tvorené označením nežiadúcej požiadavky ako neutrálnej. Záverom tejto časti testovania je fakt, že rozšírenie BlocKit vracia 1,52-krát viac správnych odhadov ako konkurenčné riešenie wyebadblock, pričom dosahuje 100 % zhodu s referenčným riešením adblock-rust.

Tabuľka 6.1: Výsledky testovania rozšírenia BlocKit v porovnaní s knižnicou adblock-rust.

	Zhoda	Nezhoda	False positive	False negative
EasyList a EasyPrivacy	242 945	0	0	0
Všetky zoznamy	242 945	0	0	0

Tabuľka 6.2: Výsledky testovania rozšírenia wyebadblock v porovnaní s knižnicou adblock-rust.

	Zhoda	Nezhoda	False positive	False negative
EasyList a EasyPrivacy	159 322	83 623	4 947	78 676
Všetky zoznamy	155 190	87 755	5 121	82 634

Súčasťou testu je i jednoduché porovnanie času vykonávania testu oboch rozšírení. Namiesto použitia uplynutého času (angl. elapsed real time), ktorý reprezentuje skutočne uplynutý čas medzi spustením programu a jeho ukončením, bol použitý čas CPU. Ten na rozdiel od uplynutého času meria množstvo času aktívne stráveného využívaním procesoru na výpočty. Vďaka tomu do tejto veličiny nie je započítaný čas, kedy program napr. čaká na vstup/výstup (angl. skratka I/O) alebo kým iný proces ukončí svoju prácu s procesorom. Ukážka kódu je uvedená vo výpise 6.1.

Čas CPU sa v jazyku C dá získať zavolaním funkcie `clock`, ktorá vracia počet hodinových taktov uplynutých od spustenia programu [22].

```
clock_t start = clock();
while (getline(&line, &len, f) != -1) {
    // spracovanie riadku a volanie funkcie na kontrolu poziadavky
}
clock_t end = clock();

printf("%f\n", (double)(end - start)/CLOCKS_PER_SEC);
```

Výpis 6.1: Výpočet času CPU stráveného vykonávaním cyklu while.

Kód začína uložením aktuálneho času CPU do premennej `start`. Hneď za priradením nasleduje časť kódu, ktorej dobu vykonávania chceme zaznamenať. V prípade tohto testu ide o cyklus `while` prechádzajúci každým riadkom upraveného datasetu a následným volaním funkcie na odoslanie požiadavky na server príslušného rozšírenia. Po ukončení cyklu je znova zaznamenaný aktuálny čas CPU, tentoraz do premennej `end`. Na získanie času stráveného vykonávaním cyklu sú tieto hodnoty od seba odčítané. Konvertovanie na sekundy prebieha vydelením konštantou `CLOCKS_PER_SEC`, ktorá vyjadruje počet hodinových taktov za sekundu. Táto konštanta obsahuje vždy inú hodnotu, podľa používaného procesoru. Pred delením konštantou je ešte nutné hodnotu času CPU pretypovať na typ `double` pre zachovanie desatinných miest.

Výsledky testovania sú dostupné v tabuľke 6.3.

Tabuľka 6.3: Výsledky porovnania priemerného času vykonania testovacieho programu pre obe rozšírenia.

	BlocKit	wyebadblock
EasyList a EasyPrivacy	2.167065s	7.159973s
Všetky zoznamy	2.342872s	7.285252s

Na výsledkoch testovania možno pozorovať viac ako 3-násobné zrýchlenie vo vyhodnocovaní rozšírenia BlocKit oproti konkurenčnému riešeniu. Rozdiely medzi testovaním len s 2 zoznamami oproti testovaniu so všetkými predvolenými je zanedbateľný.

6.2 Testovanie dopadu na rýchlosť prehliadania

Rozšírenia na filtrovanie obsahu pri vyhodnocovaní požiadaviek a prvkov stránky využívajú výpočtový výkon a operačnú pamäť systému nad rámec bežného prehliadania. Nežiadúce požiadavky sú ale často tvorené skriptami, obrázkami a prípadne i videami, ktorých veľkosť môže byť pomerne veľká. V prípade, že je požiadavka zablokovaná, prehliadač ušetrí výpočtový výkon a čas potrebný na odoslanie, čakanie na odpoveď a následné spracovanie prijatých dát.

Pri načítaní webových stránok, ktoré neobsahujú žiadne nežiadúce prvky, je adekvátne očakávať mierne spomalenie prehliadania. Naopak, pri navštívení webových stránok zaťažených väčším počtom reklám a sledovacích prvkov, je možné potenciálne zrýchlenie načítania stránky. Snahou tohto testu je zistiť, aký vplyv má rozšírenie BlocKit na rýchlosť načítania webových stránok oproti prehliadaniu bez akéhokoľvek rozšírenia.

Rýchlosť načítavania webových stránok je nedeterministická. Ovplyvňuje ju napr. nestabilná rýchlosť internetového pripojenia, či momentálne zaťaženie kontaktovaného serveru resp. vplyv rate limitu. Pri opakovaných načítaniach tej istej stránky sa odoslané požiadavky často krát líšia, čo môže byť spôsobené komplexnejšími riešeniami na zobrazovanie reklám. To všetko spôsobuje fluktuáciu rýchlosti načítania webových zdrojov, ktorá negatívne ovplyvňuje presnosť výsledkov testovania.

V snahe tento vplyv minimalizovať, som pristúpil k alternatívnemu spôsobu testovania s použitím súborov formátu HAR. Súborový formát HAR slúži na archivovanie vykonaných HTTP transakcií. Je založený na formáte JSON, čo umožňuje jednoduché spracovanie. Väčšina webových prehliadačov, vrátane tých založených na technológii WebKitGTK, umožňuje pomocou nástrojov pre vývojárov exportovať priebeh načítania stránky do súboru HAR.

Súbor uchováva informácie o každej odoslanej požiadavke, vrátane doby uplynulej od jej odoslania po prijatie v milisekundách a taktiež obsahu odpovede. Tieto údaje je možné využiť na simulovanie priebehu načítania konkrétnej webovej stránky spolu s adekvátnou dobou čakania na jednotlivé zdroje, ktorá medzi jednotlivými načítaniami stránky kolíše len minimálne.

Ako najvhodnejšie riešenie sa javilo simulátor, resp. prehrávač HAR súborov, koncipovať ako proxy server, ktorý by na požiadavky reagoval vrátením obsahu odpovede danej požiadavky z HAR súboru. V testovanom prehliadači by potom stačilo nastaviť adresu lokálne bežiaceho serveru ako adresu použitého proxy serveru.

Prehrávač som sa rozhodol implementovať v programovacom jazyku Go, kvôli jednoduchosti použitia v oblasti sieťových aplikácií. Na vytvorenie funkcionality proxy serveru bola použitá knižnica goproxy[2]. Spracovanie dodaných súborov prebieha s použitím funkcii štandardnej knižnice a následnom uložení do hašovacích tabuliek pre rýchly prístup. Prehrávač na požiadavky, ktoré nie je schopný nájsť v dodaných HAR súboroch, odpovedá stavovým kódom `404 Not found`.

Samotné testovanie prebieha opakovaným navštívením danej webovej stránky (10 načítaní) a následným výpočtom priemernej doby načítania stránky pomocou aritmetického priemeru. Odpočet doby strávenej načítavaním začína vyvolaním udalosti označenej signálom `WEBKIT_LOAD_STARTED` a končí použitím signálu `WEBKIT_LOAD_FINISHED`.

Proxy server nie je za bežných okolností schopný zachytávať šifrovanú HTTPS komunikáciu. To je možné vyriešiť nastavením vykonávania MITM (Man in the middle) útoku v prostredí knižnice goproxy. WebKitGTK ale z bezpečnostných dôvodov nedovoľuje navštíviť webovú stránku podpísanú neznámym certifikátom. Po problémoch s nastavením dôvery pre neznámy certifikát som sa rozhodol prehliadač donútiť ignorovať chyby týkajúce sa TLS pomocou funkcie `webkit_web_context_set_tls_errors_policy`.

V snahe reflektovať reálne prehliadanie internetu boli testované stránky vybrané z rebríčka 50 najnavštevovanejších webových stránok podľa spoločnosti Alexa[3]. Zároveň vybrané stránky spadajú do 3 kategórií podľa množstva nežiadúcich prvkov, ktoré boli zablokované pri načítaní v prehliadači Firefox s aktívnym rozšírením uBlock Origin s predvolenými nastaveniami filtrovacích zoznamov.

- `www.wikipedia.org` – Stránka bez akýchkoľvek nežiadúcich prvkov.
- `www.youtube.com` – Stránka s priemerným počtom nežiadúcich prvkov.
- `www.reddit.com` – Stránka s veľkým množstvom nežiadúcich prvkov.

Výsledky testovania v tabuľke 6.4 uvádzajú zanedbateľné spomalenie pri stránkach bez nežiadúcich prvkov, resp. s priemerným počtom nežiadúcich prvkov. Pri stránke s väčším počtom zablokovaných prvkov dochádza k miernemu zrýchleniu načítania stránky s rozdielom viac ako pol sekundy. Záverom testovania je, že rozšírenie BlocKit nemá vo väčšine prípadov žiadny zásadný negatívny vplyv na rýchlosť prehliadania a v určitých prípadoch ho môže urýchliť.

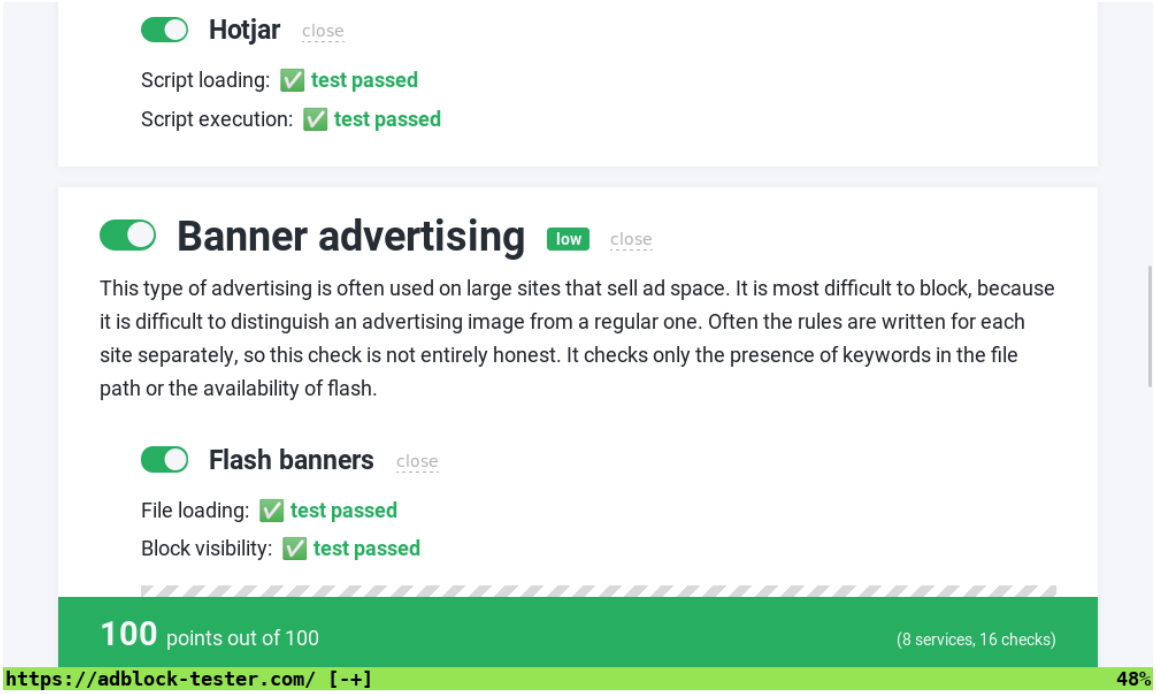
[2] https://github.com/elazarl/goproxy
[3] https://www.alexa.com/topsites

Tabuľka 6.4: Výsledky porovnania priemerného času vykonania testovacieho programu pre obe rozšírenia.

	S rozšírením BlocKit	Bez rozšírenia
Reddit	15.800513s	16.437981s
Youtube	3.862824s	3.828309s
Wikipedia	0.823924s	0.815829s

6.3 Testovacia stránka AdBlock Tester

Rozšírenia určené na blokovanie požiadaviek a reklám je taktiež možné testovať pomocou už existujúcich nástrojov. Jedným z nich je AdBlock Tester dostupný na adrese https://adblock-tester.com/. Jeho výhodou je dostupnosť v podobe jednoduchej webovej aplikácie, ktorá nevyžaduje manuálnu úpravu a vyňatie blokovacieho jadra testovaných rozšírení a umiestnenie do jednotnej testovacej sady. Aplikácia má taktiež otvorený zdrojový kód[4]. Snímka obrazovky zachytávajúca výsledky testovania stránky AdBlock Tester je dostupná formou obrázku 6.1.



Obr. 6.1: Snímka obrazovky prehliadača vimb zobrazujúca výsledky testovania na stránke https://adblock-tester.com/.

Po otvorení a načítaní stránky AdBlock Tester dôjde k ohodnoteniu blokovacím schopnostiam použitého klienta. Výsledok je prezentovaný formou počtu získaných bodov. Body sú získavané za vyhovenie testom, pričom ich je možné získať maximálne 100.

Aplikácia pri testovaní využíva reálne používané nežiadúce skripty a techniky ako sú:

[4] https://github.com/ymatuhin/adblock-tester

- Kontextová reklama
  - Google AdSense – Služba spoločnosti Google umožňujúca pridanie reklamného priestoru na stránky tretích strán výmenou za podiel na zisku. Zameranie reklám sa mení podľa toho na akej stránke sú umiestnené a komu sa práve zobrazujú (napr. podľa geografickej oblasti).
- Analytické nástroje
  - Google Analytics – Služba spoločnosti Google, ktorá umožňuje zber štatistických údajov o návštevníkoch danej webovej stránky.
  - Hotjar[5] – Nástroj slúžiaci na zaznamenávanie pohybu užívateľov po stránke. Získané dáta je možné prehrávať alebo zobraziť vo forme teplotnej mapy.
- Monitorovanie chýb
  - Sentry
  - Bugsnag
- Reklamné bannery
  - Flash banner
  - Obrázok typu GIF
  - Statický obrázok

Spolu je teda testovaných 8 služieb, pričom je každá z nich podrobená testu na načítanie skriptu/súboru a následne testu na spustiteľnosť/viditeľnosť. Výsledky testovania rozšírenia BlocKit a ďalších konkurenčných/alternatívnych rozšírení, sú dostupné v tabuľke 6.5. Pri testovaní boli použité len 2 filtrovacie zoznamy – EasyList a EasyPrivacy. Keďže pri ich súčasnom aplikovaní dosahuje referenčné riešenie uBlock Origin maximálny možný počet získaných bodov, nemá zmysel použiť ďalšie zoznamy.

Tabuľka 6.5: Počet získaných bodov pri použití jednotlivých filtrovacích zoznamov.

	BlocKit	wyebadblock	uBlock Origin
Bez zoznamu	23	23	23
EasyList	74	69	74
EasyPrivacy	55	44	55
EasyList a EasyPrivacy	100	88	100

Prehliadač Brave bol z testovania vynechaný, pretože jeho užívateľské rozhranie neumožňuje granulárne odstránenie/deaktivovanie filtrovacích zoznamov. Z jedného z konfiguračných súborov prehliadača Brave[6] je možné vyčítať, že ako jeho predvolené zoznamy sú použité predvolené zoznamy rozšírenia uBlock Origin, ktoré zahŕňajú zoznamy EasyList a EasyPrivacy. Prehliadač Brave pri navštívení testovacej stránky dosahuje skóre 100 bodov. Okrem toho rozšírenie BlocKit využíva tú istú knižnicu na blokovanie požiadaviek ako

[5] https://www.hotjar.com/

[6] https://github.com/brave/adblock-resources/blob/master/filter_lists/default.json

tento prehliadač. Podľa ostatných výsledkov testovania je teda možné s vysokou pravdepodobnosťou predpokladať, že výsledky testovania prehliadača Brave by boli rovnaké ako pri rozšírení BlocKit či uBlock Origin.

Na výsledkoch testovania v tabuľke 6.5 možno pozorovať, že rozšírenie BlocKit dosahuje pri testovaní všetkých kombinácií aktivovaných filtrovacích zoznamov rovnaké bodové ohodnotenie ako referenčné riešenie uBlock Origin. Okrem toho je adekvátne očakávať rovnaké výsledky pre prehliadač Brave, ktorý kvôli vyššie uvedeným dôvodom nebol zaradený do testovania. Konkurenčné riešenie wyebadblock naopak dosahuje len časť očakávaného počtu bodov. Pri použití oboch filtrovacích zoznamov rozšírenie nie je schopné zablokovať skript Google Analytics a taktiež nedokáže správne odstrániť Flash banner.

Tento typ testovania bol často používaný v skoršej fázy vývoja na otestovanie práve implementovanej základnej funkcionality rozšírenia.

6.4 Užívateľská odozva

Odozva od užívateľov je dôležitou súčasťou testovania, najmä pri projektoch zameraných na riešenie problému skupiny ľudí. Ešte pred začatím práce na implementácii rozšírenia som sa formou Github Issues pokúsil kontaktovať vývojárov jedného z najpoužívanejších WebKitGTK prehliadačov – Nyxt, s otázkou ohľadom kompatibility môjho rozšírenia s týmto prehliadačom[7]. Po zverejnení prvej funkčnej verzie rozšírenia na platformu Github[8], som ich užívateľov kontaktoval opäť. Podobným spôsobom som o existencii riešenia BlocKit informoval vybrané komunity na sociálnej sieti Reddit. Výsledkom bolo zvýšenie počtu stiahnutí rozšírenia, ale predovšetkým som získal cennú spätnú väzbu. Doposiaľ boli užívateľmi nahlásené 4 problémy, ktoré som následne vyriešil.

[7] https://github.com/atlas-engineer/nyxt/issues/32#issuecomment-711134676
[8] https://github.com/dudik/blockit

Kapitola 7

Záver

Cieľom tejto práce bolo navrhnúť a implementovať rozšírenie určené na blokovanie nežiadúcich požiadaviek a prvkov stránky pre prehliadače založené na technológii WebKitGTK.

Výsledkom práce je plnohodnotné rozšírenie určené na filtrovanie obsahu podporujúce okrem sieťového, aj dynamické kozmetické filtrovanie. Súčasťou je i minimalistické GUI na jednoduchú konfiguráciu a interakciu s rozšírením.

V rámci práce som sa najprv oboznámil s filtrovacími pravidlami, s ich syntaxou a s filtrovacími zoznamami, ktoré sú týmito pravidlami tvorené. Oboznámil som sa s WebKitGTK API a so spôsobom, akým v nej písať rozšírenia. Porovnal som výhody a nevýhody existujúcich riešení. Navrhol som filtrovacie rozšírenie, no počas implementácie som narazil na problém, po ktorého vyriešení sa mi naskytla možnosť od pôvodného návrhu odstúpiť a riešenie prerobiť s využitím knižnice vyvíjanej tímom stojacim za prehliadačom Brave. Síce som prišiel o určité množstvo času, no výsledné riešenie je vďaka tomuto rozhodnutiu stabilnejšie a jeho uplatnenie je do budúcnosti perspektívnejšie.

Rozšírenie bolo taktiež otestované v 3 nezávislých testoch. Prvý test sa zameral na otestovanie korektnosti a rozsahu filtrovania oproti použitej knižnici a konkurenčnému riešeniu. Výsledkom je 100 % zhoda s referenčným riešením adblock-rust, pričom konkurenčné riešenie wyebadblock dosahuje len 65.5 % zhodu a je viac ako 3-krát pomalšie. V základnom teste s použitím nástroja tretej strany rozšírenie dosiahlo opäť 100 % výsledok. Po overení, že je výsledné rozšírenie plne funkčné a dosahuje lepšie výsledky ako druhé dostupné rozšírenie, bolo nutné otestovať, aký má jeho použitie vplyv na rýchlosť prehliadania. Pri stránkach s malým až stredným počtom nežiadúcich prvkov dochádza k zanedbateľnému spomaleniu, no pri ich veľkom množstve sa prehliadanie zrýchli o viac ako pol sekundy.

Už od začiatku vývoja prebiehala komunikácia s potenciálnou cieľovou užívateľskou skupinou, vďaka ktorej boli nájdené a opravené 4 chyby. Riešenie je uverejnené na platforme Github, kde od užívateľov získalo 33 hviezd, pričom si ho doposiaľ stiahli viac ako 140-krát.

Vďaka práci som si mohol prakticky vyskúšať využitie medzi-procesovej komunikácie na riešenie netriviálnych problémov. Okrem toho som prišiel do styku s 2, pre mňa doposiaľ neznámymi programovacími jazykmi, pričom oba plánujem používať i v budúcnosti.

Zámer práce bol splnený, no ďalej by som sa chcel venovať miernej refaktorizácii kódu za účelom zjednodušenia budúcich úprav a taktiež pridaniu funkcionality umožňujúcej jednoduchú diagnostiku a ladenie filtrovacích pravidiel, resp. zoznamov. Obzvlášť by som chcel spomenúť testovanie dopadu na rýchlosť prehliadania, ktoré by po rozšírení mohlo slúžiť na deterministické testovanie filtrovacích rozšírení všetkých dostupných prehliadačov.

Literatúra

[1] Aina, E. *Port the GNOME Web adblocker to the Content Blockers API* [online]. November 2015 [cit. 20.04.2021]. Dostupné z: https://gitlab.gnome.org/GNOME/epiphany/-/issues/288.

[2] Alphabet. *Alphabet Announces Third Quarter2020 Results* [online]. Október 2020 [cit. 30.11.2020]. Dostupné z: https://abc.xyz/investor/static/pdf/2020Q3_alphabet_earnings_release.pdf?cache=514fb58.

[3] Bloom, B. H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM* [online]. 1. vyd. Júl 1970, zv. 13, č. 7, s. 422–426, [cit. 03.05.2021]. DOI: 10.1145/362686.362692. Dostupné z: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.641.9096&rep=rep1&type=pdf.

[4] Brave. *Brave Improves Its Ad-Blocker Performance by 69x with New Engine Implementation in Rust* [online]. Jún 2019 [cit. 22.12.2021]. Dostupné z: https://brave.com/improved-ad-blocker-performance/.

[5] Campos, C. G. *WebKit2GTK+ Web Process Extensions* [online]. September 2013 [cit. 03.01.2021]. Dostupné z: https://blogs.igalia.com/carlosgc/2013/09/10/webkit2gtk-web-process-extensions/.

[6] Docs, M. W. *Content-Security-Policy* [online]. Február 2021 [cit. 21.03.2021]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy.

[7] Docs, M. W. *WebRequest.ResourceType* [online]. Február 2021 [cit. 21.03.2021]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/ResourceType.

[8] Eilertsen, I. K. *Syntax meanings that are actually human readable* [online]. Január 2021 [cit. 13.01.2021]. Dostupné z: https://github.com/DandelionSprout/adfilt/blob/master/Wiki/SyntaxMeaningsThatAreActuallyHumanReadable.md.

[9] eye/o. *How to write filters* [online]. 2020 [cit. 13.01.2021]. Dostupné z: https://help.eyeo.com/adblockplus/how-to-write-filters.

[10] Gerard, D. *Brave web browser no longer claims to fundraise on behalf of others — so that's nice* [online]. Január 2019 [cit. 10.12.2020]. Dostupné z: https://davidgerard.co.uk/blockchain/2019/01/13/brave-web-browser-no-longer-claims-to-fundraise-on-behalf-of-others-so-thats-nice/.

[11] GTK Doc. *WebKitGTK Reference Manual* [online]. 2021 [cit. 21.04.2021]. Dostupné z: https://webkitgtk.org/reference/webkit2gtk/stable/index.html.

[12] Hill, R. *ABP's $rewrite filter option* [online]. Máj 2018 [cit. 13.01.2021]. Dostupné z: https://github.com/uBlockOrigin/uBlock-issues/issues/46.

[13] Hill, R. *Ublock orgin readme.md* [online]. Február 2021 [cit. 10.12.2020]. Dostupné z: https://github.com/gorhill/uBlock/blob/master/README.md.

[14] iTDM. *Mozilla Firefox Usage Down 85% but why are Exec's Salary Up 400%?* [online]. September 2020 [cit. 30.11.2020]. Dostupné z: https://itdm.com/mozilla-firefox-usage-down-85-but-why-are-execs-salary-up-400/2050/.

[15] Keck, C. *Apple Seems to Be Tracking iPhone 11 When Location Services Are Disabled, Report Finds* [online]. Apríl 2019 [cit. 30.11.2020]. Dostupné z: https://gizmodo.com/apple-seems-to-be-tracking-iphone-11-when-location-serv-1840204086.

[16] Lindsey, N. *Google Blocking Web Privacy Proposals at W3C* [online]. Október 2019 [cit. 30.11.2020]. Dostupné z: https://www.cpomagazine.com/data-privacy/google-blocking-web-privacy-proposals-at-w3c/.

[17] Lyons, K. *Brave browser CEO apologizes for automatically adding affiliate links to cryptocurrency URLs* [online]. Jún 2020 [cit. 10.12.2020]. Dostupné z: https://www.theverge.com/2020/6/8/21283769/brave-browser-affiliate-links-crypto-privacy-ceo-apology.

[18] McIlroy, D. *Basics of the Unix Philosophy* [online]. 1978 [cit. 20.04.2021]. Dostupné z: http://www.catb.org/esr/writings/taoup/html/ch01s06.html.

[19] Palant, W. *Acceptable ads in Adblock Plus* [online]. December 2011 [cit. 10.12.2020]. Dostupné z: https://adblockplus.org/development-builds/allowing-acceptable-ads-in-adblock-plus.

[20] Plus, A. *Adblock Plus filters explained* [online]. 2020 [cit. 13.01.2021]. Dostupné z: https://adblockplus.org/filter-cheatsheet.

[21] Poulain, B. *Introduction to WebKit Content Blockers* [online]. Jún 2015 [cit. 22.01.2021]. Dostupné z: https://webkit.org/blog/3476/content-blockers-first-look/.

[22] project, T. G. *CPU Time (The GNU C Library)* [online]. 2021 [cit. 20.04.2021]. Dostupné z: https://www.gnu.org/software/libc/manual/html_node/CPU-Time.html.

[23] statcounter. *Browser Market Share Worldwide* [online]. 2020 [cit. 30.11.2020]. Dostupné z: https://gs.statcounter.com/browser-market-share.

[24] team, P. *Adblockers Performance Study* [online]. Február 2019 [cit. 21.04.2021]. Dostupné z: https://whotracks.me/blog/adblockers_performance_study.html.

[25] WebKitGTK. *WebkitGTK introduction* [online]. 2019 [cit. 28.04.2021]. Dostupné z: https://webkitgtk.org/.

Príloha A

Obsah priloženého pamäťového média

- `/`
  - `virtual.ova` – Súbor virtuálneho stroja demonštrujúci výsledné riešenie.
  - `demonstration.mp4` – Demonštračné video vytvorené pre konferenciu Excel@FIT.
  - `thesis.pdf` – Technická správa.
  - `thesis-source` – Zdrojový kód technickej správy.
  - `analysis` – Python skripty použité na analýzu z kapitoly 2.2.
  - `tests` – Testovacie programy z kapitoly 6.
  - `blockit` – Zdrojové súbory rozšírenia.
    - `docs` – Doxygen dokumentácia.
    - `README.md` – Informácie o rozšírení, postup inštalácie a návod na použitie.
  - `adblock-rust-server` – Zdrojové súbory filtrovacieho serveru.
    - `docs` – rustdoc dokumentácia.
    - `README.md` – Informácie o filtrovacom serveri.
  - `old-adblock` – Zdrojové súbory pôvodnej verzie blokovacieho jadra, ktoré bolo neskôr nahradené serverom.

Virtuálny stroj

Súbor virtuálneho stroja `virtual.ova` obsahuje operačný systém Alpine Linux so správcom okien Openbox. Táto kombinácia bola zvolená v snahe minimalizovať veľkosť výsledného súboru. Po spustení je automaticky otvorené okno prehliadača vimb so stránkou Adblock Tester bez nutnosti prihlasovania do systému alebo akejkoľvek inej interakcie. Navigácia v prehliadači je možná stlačením klávesy písmena `o`. V prehliadači je automaticky aktivované rozšírenie BlocKit s filtrami EasyList a EasyPrivacy.

Príloha B

Plagát

Obr. B.1: Plagát na Excel@FIT 2021.